aws

Guide du développeur

Amazon DynamoDB

Version de l'API 2012-08-10

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Guide du développeur

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Les marques et la présentation commerciale d’Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n’est pas d’Amazon, d’une manière susceptible de créer une confusion parmi les clients, ou d’une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon sont la propriété de leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce qu'Amazon DynamoDB ? 1
Caractéristiques 2
sans serveur 2
NoSQL 2
Entièrement géré 2
Performances inférieures à 10 millisecondes, quelle que soit l'échelle 3
Cas d'utilisation 3
Fonctionnalités 4
Réplication multiactive avec les tables globales 4
Transactions ACID 5
Capture des données modifiées 5
Index secondaires 5
Intégrations de service 5
Intégrations sans serveur 6
Importation et exportation de données vers Amazon S3 6
Intégration zéro ETL 6
Mise en cache 6
Sécurité 7
Résilience 8
Tables globales 8
Sauvegardes et point-in-time restauration continues 8
Sauvegarde et restauration à la demande 9
Accès à DynamoDB 9
Tarification 9
Prise en main 9
Mise en route avec DynamoDB 11
Ressources pour les nouveaux utilisateurs 12
Meilleures pratiques supplémentaires d'Amazon DynamoDB pour les nouveaux utilisateurs 13
Ressources AWS CLI 13
Ressources de programmation 14
Accès à DynamoDB 14
Utilisation de la console 15
À l'aide du AWS CLI 15

Utilisation de l'API 18
Utilisation de NoSQL Workbench 18
Plages d'adresses IP 20
Points de terminaison à double pile pour le protocole Internet version 6 () IPv6 20
Conditions préalables 21
Configuration de DynamoDB 21
Configuration de DynamoDB (service web) 22
Configuration de DynamoDB Local (version téléchargeable) 26
Étape 1 : créer une table 54
Étape 2 : écrire des données 108
Étape 3 : lire des données 140
Étape 4 : mettre à jour des données 168
Étape 5 : interroger des données 201
Étape 6 : nettoyage (facultatif) 240
Étapes suivantes 259
Console-to-Code 259
Comment ça marche 260
Avantages de l'utilisation Console-to-Code avec DynamoDB 261
Exemples de cas d'utilisation 261
Prise en main 261
Comment ça marche 263
Aide-mémoire 263
Configuration initiale du 263
SDK ou CLI 264
Actions de base 264
Règles de dénomination 265
Principes de base des quotas de service 266
En savoir plus 268
Composants de base 268
Tables, éléments et attributs 269
Clé primaire 273
Index secondaires 274
DynamoDB Streams 277
API DynamoDB 279
Plan de contrôle 279
Plan de données 280

DynamoDB Streams ... 282
Transactions ... 282
Types de données et règles de dénomination pris en charge ... 283
Règles de dénomination ... 283
Types de données ... 284
Descripteurs de type de données ... 290
Classes de tables DynamoDB ... 290
Partitions et distribution des données dans DynamoDB ... 291
Distribution de données : clé de partition ... 292
Distribution de données : clé de partition et clé de tri ... 293
Découvrez comment passer de SQL à NoSQL ... 296
Système relationnel ou NoSQL ? ... 297
Accès et authentification ... 300
Création d'une table ... 304
Obtention d'informations sur une table ... 306
Écriture de données dans une table ... 308
Lecture de données à partir d'une table ... 312
Gestion des index ... 321
Modification de données dans une table ... 327
Suppression de données d'une table ... 331
Suppression d'une table ... 333
Ressources et outils de formation Amazon DynamoDB ... 334
Outils de codage et de visualisation ... 334
Conseils prescriptifs ... 335
Centre de connaissances ... 336
Articles de blog, référentiels et guides ... 337
Modélisation des données et modèles de conception ... 337
Formation ... 338
Lectures et écritures ... 339
Cohérence en lecture DynamoDB ... 339
Lectures cohérentes à terme ... 339
Lectures fortement cohérentes ... 340
Cohérence de lecture des tables globales ... 340
Opérations de lecture et d'écriture ... 340
Consommation d'opérations de lecture ... 341
Consommation d'opérations d'écriture ... 342

Capacité de débit DynamoDB 345
Mode de capacité à la demande 345
Mode alloué 346
Mode de capacité à la demande DynamoDB 346
Unités de demande de lecture et unités de demande d'écriture 348
Débit initial et propriétés de mise à l'échelle 348
Débit maximal DynamoDB pour les tables à la demande 349
Mode de capacité provisionnée 351
Unités de capacité de lecture et d'écriture 353
Choix des paramètres de débit initial 353
Mise à l'échelle automatique de DynamoDB 354
Gestion de la capacité de débit avec la mise à l'échelle automatique 355
Capacité réservée 387
Débit chaud 388
Vérification du débit chaud de votre table 389
Augmentation du débit chaud de votre table 392
Création d'une table avec un débit chaud supérieur 399
Scénarios de débit chaud 410
Débordement et capacité adaptative 412
Capacité de débordement 412
Capacité adaptative 413
Changement de mode de capacité 414
Mode provisionné vers mode à la demande 415
Mode à la demande vers mode provisionné 417
Programmation avec DynamoDB 419
Présentation de la prise en charge du AWS SDK pour DynamoDB 419
Support du SDK pour les points de terminaison basés sur des AWS comptes 421
Interfaces de programmation compatibles avec DynamoDB 422
Interfaces de programmation de niveau supérieur 430
Exécution des exemples de code 493
API de bas niveau 500
Programmation avec Python 506
À propos de Boto 507
Documentation Boto 507
Couches du client et des ressources 508
Utilisation de batch_writer 511

Exemples supplémentaires de code 512
Sécurité des sessions et des threads 513
Config 513
Gestion des erreurs 518
Journalisation 521
Hooks d'événement 522
Pagination et paginateur 523
Programmes d'attente 525
Programmation avec JavaScript 526
À propos de AWS SDK pour JavaScript 526
AWS SDK pour JavaScript V3 527
Documentation JavaScript 527
Couches d'abstraction 528
Fonction utilitaire de regroupement 530
Lecture d'éléments 531
Écritures conditionnelles 532
Pagination 533
Config 535
Programmes d'attente 538
Gestion des erreurs 539
Journalisation 541
Considérations 542
Programmation avec le AWS SDK for Java 2.x 543
À propos de AWS SDK for Java 2.x 544
Prise en main 545
Documentation du Kit SDK pour Java 2.x 554
Interfaces prises en charge 555
Exemples supplémentaires de code 570
Programmation synchrone et asynchrone 570
Clients HTTP 571
Config 572
Gestion des erreurs 580
AWS ID de demande 581
Logging 581
Pagination 584
Annotations de classes de données 585

Gestion des erreurs ... 586
Composants erreur ... 586
Erreurs transactionnelles ... 587
Codes et messages d'erreur ... 587
Gestion des erreurs dans votre application ... 593
Nouvelles tentatives après erreur et backoff exponentiel ... 593
Opérations par lot et gestion des erreurs ... 594
Travailler avec AWS SDKs ... 595
Utilisation de DynamoDB ... 597
Utilisation de tables ... 597
Opérations de base sur les tables ... 598
Points à prendre en considération lors du choix d'une classe de tables dans DynamoDB ... 607
Balises et étiquettes ... 608
Utilisation de tables globales ... 614
Comment ça marche ... 615
Tutoriels : Création de tables globales ... 625
Sécurité des tables globales ... 663
Facturation des tables globales ... 676
Versions des tables globales ... 679
Bonnes pratiques relatives aux tables globales ... 691
Utilisation d'éléments ... 693
Tailles et formats d'élément ... 694
Lecture d'un élément ... 695
Écriture d'un élément ... 696
Valeurs renvoyées ... 699
Opérations par lots ... 700
Compteurs atomiques ... 703
Écritures conditionnelles ... 704
Utilisation d'expressions ... 710
Durée de vie (TTL) ... 752
Interrogation de tables ... 783
Analyse de tables ... 793
Langage de requête PartiQL ... 802
Utilisation d'éléments : Java ... 851
Utilisation d'élements : .NET ... 883
Utilisation des index ... 917

Index secondaires globaux 922
Index locaux secondaires 1022
Utilisation des transactions 1077
Comment ça marche 1078
Utilisation d'IAM avec des transactions 1088
Exemple de code 1091
Utilisation des flux 1095
Options 1096
Utilisation de Kinesis Data Streams 1098
Utilisation de DynamoDB Streams 1117
Accélération en mémoire avec DAX 1201
Cas d'utilisation pour DAX 1202
Notes d'utilisation de DAX 1203
Comment ça marche 1204
Comment DAX traite les demandes 1206
Cache d'élément 1208
Cache de requête 1209
Composants de cluster DAX 1210
Nœuds 1210
Clusters 1211
Régions et Zones de disponibilité 1212
Groupes de paramètres 1213
Groupes de sécurité 1213
ARN de cluster 1214
Point de terminaison de cluster 1214
Points de terminaison de nœud 1214
Groupes de sous-réseaux 1215
Événements 1215
Fenêtre de maintenance 1215
Création d'un cluster DAX 1217
Création d'une fonction du service IAM pour permettre à DAX d'accéder à DynamoDB 1217
DAX et IPv6 1219
À l'aide du AWS CLI 1222
Utilisation de la console 1228
Modèles de cohérence 1234
Cohérence entre les nœuds d'un cluster DAX 1234

Comportement du cache d'éléments DAX 1235
Comportement du cache de requêtes DAX 1238
Lectures transactionnelles à cohérence forte 1239
Mise en cache négative 1240
Politiques pour les écritures 1240
Développement avec le client DAX 1244
Didacticiel : exécution d'un exemple d'application 1244
Modification d'une application existante pour utiliser DAX 1311
Gestion des clusters DAX 1312
Autorisations IAM pour gérer un cluster DAX 1312
Mise à l'échelle d'un cluster DAX 1315
Personnalisation des paramètres de cluster DAX 1317
Configuration des paramètres de durée de vie (TTL) 1318
Prise en charge de l'étiquetage pour DAX 1320
AWS CloudTrail intégration 1321
Suppression d'un cluster DAX 1321
Surveillance de l'accélérateur DynamoDB 1321
Outils de surveillance pour DAX 1322
Surveillance avec CloudWatch 1324
Journalisation des opérations DAX avec AWS CloudTrail 1351
Instances extensibles DAX T3/T2 1352
Famille d'instances DAX T2 1352
Famille d'instances DAX T3 1352
Contrôle d'accès à DAX 1353
Fonction du service IAM pour DAX 1354
Politique IAM pour autoriser l'accès au cluster DAX 1356
Étude de cas : accès à DynamoDB et à DAX 1357
Accès à DynamoDB, mais pas d'accès avec DAX 1359
Accès à DynamoDB et à DAX 1361
Accès à DynamoDB via DAX, mais aucun accès direct à DynamoDB 1367
Chiffrement au repos DAX 1369
Activation du chiffrement au repos à l'aide de la AWS Management Console 1372
Chiffrement DAX en transit 1373
Utilisation des rôles liés à un service pour DAX 1373
Autorisations des rôles liés à un service pour DAX 1374
Création d'un rôle lié à un service pour DAX 1376

Modification d'un rôle lié à un service pour DAX ... 1376
Suppression d'un rôle lié à un service pour DAX ... 1376
Accès au DAX sur plusieurs comptes AWS ... 1378
Configurer IAM ... 1378
Configuration d’un VPC ... 1381
Modifier le client DAX pour autoriser l’accès intercompte ... 1384
Guide de dimensionnement de cluster DAX ... 1389
Présentation ... 1389
Estimation du trafic ... 1389
Test de charge ... 1390
Modélisation des données ... 1393
Travailler avec les collections d'éléments ... 1395
Accélérer les requêtes en organisant vos données avec des collections d'articles ... 1396
Principes de base de la modélisation des données ... 1396
Conception à une seule table ... 1397
Conception à plusieurs tables ... 1400
Composantes de base de la modélisation des données ... 1402
Clé de tri composite ... 1402
Multilocataire ... 1404
Index fragmenté ... 1405
Durée de vie ... 1406
Durée de vie pour l’archivage ... 1408
Partitionnement vertical ... 1409
Partitionnement d’écriture ... 1412
Packages de conception de schémas de modélisation de données ... 1413
Prérequis ... 1414
Réseau social ... 1415
Profil de jeu ... 1424
Système de gestion des réclamations ... 1433
Paiements récurrents ... 1452
Mises à jour du statut d’un appareil ... 1458
Boutique en ligne ... 1471
Modélisation relationnelle ... 1495
Modèles de base de données relationnelle traditionnels ... 1496
Comment DynamoDB élimine le besoin d’opérations JOIN ... 1498
Comment les transactions DynamoDB éliminent la surcharge pour le processus d’écriture 1499

Premiers pas 1500
Exemple 1502
Migration vers DynamoDB 1508
Raisons de la migration 1508
Considérations relatives à la migration 1510
Comment ça marche 1512
Outils de migration 1513
Choix d'une stratégie de migration 1514
Migration hors ligne 1516
Migration hybride 1518
En ligne : migration de chaque table 1:1 1520
En ligne : migration à l'aide d'une table intermédiaire personnalisée 1521
NoSQL Workbench 1524
Téléchargement 1525
Installation 1527
Modélisateur de données 1534
Création d'un modèle 1534
Importation d'un modèle existant 1541
Exportation d'un modèle 1544
Modification d'un modèle existant 1546
Visualiseur de données 1550
Ajout d'un exemple de données 1550
Importation à partir de CVS 1553
Facettes 1554
Vue d'agrégation 1557
Validation d'un modèle de données 1558
Créateur d'opérations 1561
Se connecter à des ensembles de données 1562
Création d'opérations 1563
Clonage de tables 1575
Exportation vers CSV 1577
Exemples de modèle de données 1578
Modèle de données d'employé 1578
Modèle de données de forum de discussion 1579
Modèle de données de bibliothèque musicale 1579
Modèle de données de station de ski 1580

Modèle de données d'offres de carte de crédit 1580
Modèle de données de signets 1581
Historique de versions 1581
Sauvegarde et restauration 1589
Sauvegardes ponctuelles 1591
Avant de commencer 1591
Activation de la reprise ponctuelle 1592
Sauvegardes à la demande 1597
Comment ça marche 1597
Sauvegarde d’une table 1601
Restauration d’une table 1603
Suppression d'une sauvegarde de table 1609
Utilisation d’IAM 1610
Facturation des sauvegardes 1618
Comment ça marche 1618
Exemple de facturation de sauvegarde DynamoDB 1619
Restaurations 1624
Restauration d’une table à l’aide de la reprise ponctuelle 1624
Restauration d’une table DynamoDB à un instant dans le passé 1626
Utilisation de AWS Backup 1631
Comment ça marche 1633
Création de sauvegardes 1636
Copie de sauvegardes 1638
Restauration d'une table 1639
Suppression de sauvegardes 1640
Sauvegardes à la demande gérées par AWS Backup par rapport à DynamoDB 1641
Exemples de code 1643
Principes de base 1656
Hello DynamoDB 1657
Principes de base 1668
Actions 1825
Scénarios 2237
Accélérer les lectures avec DAX 2240
Scénarios d’index secondaire global avancés 2249
Créer une application pour soumettre des données à une table DynamoDB 2252
Comparaison de plusieurs valeurs avec un seul attribut 2253

Mise à jour sous certaines conditions du TTL d'un élément 2276
Connexion à une instance locale 2283
Comptage des opérateurs d'expression 2284
Créer une API REST pour suivre les données de la COVID-19 2317
Créer une application de messagerie 2318
Création d'une application sans serveur pour gérer des photos 2319
Création d'une table avec un index secondaire global 2323
Création d'une table avec le débit à chaud activé 2327
Créer une application web pour suivre les données DynamoDB 2337
Créer une application de chat WebSocket 2339
Création d'un élément avec un TTL 2340
Création et gestion des tables globales MRSC 2347
Création et gestion de tables globales démontrant MREC 2367
Suppression de données à l'aide de PartiQL DELETE 2377
Détecter l'EPI dans des images 2383
Insertion de données à l'aide de PartiQL INSERT 2384
Invoquer une fonction Lambda à partir d'un navigateur 2388
Gestion des index secondaires globaux 2389
Gestion des politiques basées sur les ressources 2391
Surveillance des performances DynamoDB 2394
Exécution d'opérations de requête avancées 2395
Exécution d'opérations de liste 2430
Exécution d'opérations de mappage 2463
Exécution d'opérations d'ensemble 2487
Interrogation d'une table à l'aide de lots d'instructions PartiQL 2517
Interrogation d'une table à l'aide de PartiQL 2580
Interrogation d'une table à l'aide d'un index secondaire global 2634
Interrogation d'une table à l'aide d'une condition begins_with 2643
Interrogation d'une table à l'aide d'une plage de dates 2649
Interrogation d'une table avec une expression de filtre complexe 2659
Interrogation d'une table avec une expression de filtre dynamique 2667
Interrogation d'une table avec une expression de filtre et des limites 2676
Interrogation d'une table avec des attributs imbriqués 2681
Interrogation d'une table avec pagination 2690
Interrogation d'une table avec des lectures fortement cohérentes 2701
Interrogation des données à l'aide de PartiQL SELECT 2705

Interrogation des éléments TTL 2711
Interrogation de tables à l'aide de modèles de date et d'heure 2715
Enregistrer des informations EXIF et d'autres informations sur les images 2730
Configuration d'un contrôle d'accès par attributs 2731
Compréhension de l'ordre des expressions de mise à jour 2734
Mise à jour du paramètre de débit chaud d'une table 2765
Mise à jour du TTL d'un élément 2771
Mise à jour des données à l'aide de PartiQL UPDATE 2775
Utiliser API Gateway pour invoquer une fonction Lambda 2782
Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda 2784
Utiliser un modèle de document 2784
Utiliser un modèle de persistance des objets de haut niveau 2800
Utilisation d'opérations de compteurs atomiques 2810
Utilisation d'opérations conditionnelles 2833
Utilisation des noms d'attributs d'expression 2863
Utilisent des événements planifiés pour invoquer une fonction Lambda 2891
Utilisation d'index secondaires locaux 2893
Travaillez avec Streams et Time-to-Live 2895
Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC) 2899
Utilisation du balisage des ressources 2903
Utilisation du chiffrement des tables 2906
Exemples sans serveur 2908
Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB 2908
Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB 2917
AWS contributions communautaires 2928
Création et test d'une application sans serveur 2929
Sécurité 2931
AWS politiques gérées 2932
AWS politiques gérées 2932
AWS politique gérée : AmazonDynamo DBFull Access_v2 2933
AmazonDynamoDBReadOnlyAccess 2933
Mises à jour DynamoDB des politiques gérées AWS 2935
Politiques basées sur les ressources 2937
Create table 2938

Attache d’une politique basée sur les ressources 2944
Attache d’une politique à un flux 2949
Suppression d’une politique basée sur les ressources 2952
Accès intercomptes 2953
Blocage de l’accès public 2954
opérations d’API 2957
autorisation IAM 2966
Exemples 2966
Considérations 2974
Bonnes pratiques 2975
Contrôle d'accès basé sur les attributs 2977
Pourquoi utiliser l’ABAC ? 2979
Clés de condition 2979
Considérations 2980
Activation de l’ABAC dans DynamoDB 2980
Utilisation de l’ABAC 2983
Exemples de cas d’utilisation 2985
Résolution des problèmes 2999
Protection des données 3000
Chiffrement au repos 3001
Sécurisation de connexions DynamoDB 3028
IAM 3030
Gestion de l’identité et des accès 3030
Utilisation de conditions 3063
Validation de conformité 3090
Résilience 3090
Sécurité de l’infrastructure 3091
Utilisation de points de terminaison d’un VPC 3092
AWS PrivateLink pour DynamoDB 3102
Types de points de terminaison d’un VPC Amazon 3103
Considérations relatives à l'utilisation AWS PrivateLink d'Amazon DynamoDB 3104
Création d’un point de terminaison Amazon VPC 3105
Accès aux points de terminaison de l’interface Amazon DynamoDB 3105
Accès aux tables DynamoDB et opérations d’API de contrôle depuis les points de terminaison de l’interface DynamoDB 3105
Mise à jour d’une configuration DNS sur site 3108

Création d'une politique de point de terminaison d'un VPC Amazon 3110
Utilisation de points de terminaison DynamoDB avec AWS Management Console Private Access 3111
AWS PrivateLink pour DynamoDB Streams 3112
AWS PrivateLink pour DAX 3118
Analyse de la configuration et des vulnérabilités 3121
Bonnes pratiques de sécurité 3121
Bonnes pratiques de sécurité préventive 3122
Bonnes pratiques de sécurité de détection 3125
Surveillance et journalisation 3129
Plan de surveillance 3129
Référence des performances 3129
Services intégrés 3130
Outils de surveillance automatique 3130
Surveillance des métriques 3131
Comment utiliser les métriques de DynamoDB ? 3132
Affichage des métriques dans la console CloudWatch 3133
Affichage des métriques dans la AWS CLI 3133
Dimensions et métriques 3134
Création d' CloudWatch alarmes dans DynamoDB 3166
Journalisation des opérations 3169
Informations de DynamoDB dans CloudTrail 3170
Présentation des entrées de fichier journal DynamoDB 3174
Contributor Insights 3193
Comment ça marche 3194
Premiers pas 3205
Utilisation d'IAM 3214
Bonnes pratiques 3221
Conception NoSQL 3222
NoSQL comparé au SGBDR 3222
Deux concepts essentiels 3223
Approche générale 3223
NoSQL Workbench 3224
Cadre DynamoDB Well-Architected 3225
Optimisation des coûts 3225
Réalisation de l'évaluation du cadre Well-Architected d'Amazon DynamoDB 3277

Piliers du cadre Well-Architected d'Amazon DynamoDB 3277
Création de clés de partition 3280
Répartition des charges de travail 3281
Partitionnement d'écriture 3282
Chargement efficace de données 3284
Conception de clé de tri 3286
Contrôle de version 3287
Index secondaires 3288
Consignes générales 3289
Index fragmentés 3292
Agrégation 3294
Surcharge d'index secondaire global 3295
Partitionnement d'index secondaire global 3297
Création d'un réplica 3302
Éléments volumineux 3303
Compression 3304
Partitionnement vertical 3304
Utilisation d'Amazon S3 3305
Données de séries temporelles 3305
Modèle de création des données de séries temporelles 3306
Exemples de tableaux de séries chronologiques 3306
Many-to-many relations 3307
Listes adjacentes 3308
Graphiques matérialisés 3309
Interrogation et analyse 3314
Performances d'analyse 3314
Éviter les pics 3315
Analyses parallèles 3318
Conception d'une table 3319
Utilisation de tables globales 3319
Faits importants 3320
Faits importants à propos de la MREC 3321
Faits importants à propos de la MRSC 3322
Cas d'utilisation 3324
Modes d'écriture 3325
Stratégies de routage dans DynamoDB 3332

Processus d'évacuation 3338
Planification de la capacité de débit 3342
Liste de contrôle pour la préparation 3343
Conclusion et ressources 3352
Plan de contrôle 3353
Opérations de données groupées 3354
Mise à jour de lot conditionnelle 3355
Opérations groupées efficaces 3360
Mise en œuvre du contrôle de version 3361
Quand utiliser ce modèle 3362
Conception de modèle 3363
Utilisation du motif 3363
Rapports d'utilisation et de facturation 3366
Capacité de débit 3369
Flux 3374
Stockage 3374
Sauvegarde et restauration 3375
Transfert de données 3379
CloudWatch 3380
DAX 3381
Migration d'une table DynamoDB d'un compte à un autre 3383
Migration d'une table avec AWS Backup pour la sauvegarde et la restauration entre comptes 3384
Migration d'une table à l'aide de l'exportation vers S3 et de l'importation depuis S3 3386
Recommandations prescriptives concernant DAX 3389
Évaluation de l'adéquation de DAX 3389
Configuration de votre client DAX 3392
Configuration de votre cluster DAX 3393
Dimensionnement de votre cluster DAX 3400
Déploiement d'un cluster 3407
Opérations de cluster 3409
Surveillance de DAX 3412
Utilisation de DynamoDB avec d'autres services AWS 3415
Intégration à Amazon Cognito 3415
Intégration avec Amazon Redshift 3418
Considérations relatives à l'intégration entre comptes avec CMK 3418

Intégration zéro ETL à Amazon Redshift ... 3422
Chargement de données de DynamoDB dans Amazon Redshift à l'aide de COPY ... 3431
Intégration avec Amazon EMR ... 3433
Présentation ... 3433
Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive ... 3434
Création d'une table externe dans Hive ... 3443
Traitement des instructions HiveQL ... 3447
Interrogation de données dans DynamoDB ... 3448
Copie de données depuis et vers Amazon DynamoDB ... 3451
Personnalisation de performances ... 3465
Intégration avec S3 ... 3471
Importer depuis Amazon S3 ... 3472
Exporter vers Amazon S3 ... 3493
Intégration à Amazon SageMaker Lakehouse ... 3516
Intégration zéro ETL à Amazon SageMaker Lakehouse ... 3516
Intégration à Amazon OpenSearch Service ... 3518
Fonctionnement ... 3519
Création d'une intégration ... 3520
Étapes suivantes ... 3520
Gestion des modifications avec rupture ... 3520
Intégration zéro ETL à OpenSearch Service ... 3524
Intégration à Amazon EventBridge ... 3527
Comment ça marche ... 3528
Mise à jour d'une intégration via la console ... 3529
Étapes suivantes ... 3531
Intégration à Amazon MSK ... 3532
Fonctionnement ... 3533
Exemple d'intégration ... 3533
Étapes suivantes ... 3540
Bonnes pratiques d'intégration ... 3540
Création d'un instantané ... 3540
Capture de données modifiées ... 3541
Utilisation de l'IA générative avec DynamoDB ... 3542
Cas d'utilisation de l'IA générative pour DynamoDB ... 3542
Blogs sur l'IA générative pour DynamoDB ... 3543
Tirer parti de l'intégration de DynamoDB Zero-ETL avec Service OpenSearch ... 3543

Quotas et contraintes 3545
Exécution de tâches de gestion des quotas 3545
Accès aux quotas de DynamoDB 3545
Affichage des quotas actuels dans la console 3546
Affichage des quotas actuels à l'aide de l'AWS CLI 3547
Demande d'augmentation de quota 3549
Quotas 3549
Débit de lecture/écriture 3550
Capacités réservées 387
Tables 3554
Tables globales 3554
Index secondaires 3555
Attributs d'index secondaire projetés 3555
DynamoDB Streams 3556
Importer des données depuis Amazon S3 3556
Exportation de table vers Amazon S3. 3557
Sauvegarde et restauration 3557
Contributor Insights 3557
Constaintes 3557
Mode de capacité en lecture/écriture 3558
Index secondaires 3555
Clés de partition et clés de tri 3559
Règles de dénomination 3560
Types de données 3561
Éléments 3562
Attributes 3562
Paramètre d'expression 3563
Transactions DynamoDB 3564
DynamoDB Streams 3564
DynamoDB Accelerator (DAX) 3565
Contraintes propres à l'API 3565
Chiffrement de DynamoDB au repos 3568
Référence d'API 3569
Résolution de problème 3570
Erreurs internes du serveur 3570
Examen des erreurs internes du serveur 3571

Minimisation de l'impact des erreurs internes du serveur ... 3572
Amélioration de la conscience opérationnelle ... 3572
Latence ... 3576
étranglement ... 3580
Diagnostic des problèmes de limitation ... 3582
Guide de résolution ... 3589
Limitation d'écriture et contre-pression des GSI ... 3617
Métriques de limitation CloudWatch ... 3619
Annexe ... 3622
Résolution des problèmes d'établissement de SSL/TLS connexion avec DynamoDB ... 3622
Test de votre application ou service ... 3622
Test de votre navigateur client ... 3623
Mise à jour de votre client d'application logicielle ... 3623
Mise à jour de votre navigateur client ... 3624
Mise à jour manuelle de votre solution groupée de certificats ... 3624
Exemples de tables et de données à utiliser dans DynamoDB ... 3625
Fichiers de données d'exemple ... 3626
Création d'exemples de tables et chargement de données ... 3639
Création d'exemples de tables et chargement de données – Java ... 3639
Création d'exemples de tables et chargement de données – .NET ... 3649
Exemple d'application utilisant AWS SDK pour Python (Boto3) ... 3661
Étape 1 : déploiement et test localement ... 3662
Étape 2 : examen du modèle de données et des détails de la mise en œuvre ... 3667
Étape 3 : déploiement en production ... 3677
Étape 4 : Nettoyer les ressources ... 3687
Mots réservés dans DynamoDB ... 3687
AWS Exemples de SDK pour Java 1.x ... 3700
DAX et kit SDK Java v1 ... 3701
Modification d'une application du kit SDK pour Java 1.x pour utiliser DAX ... 3713
Interrogation d'index secondaires globaux avec le kit SDK pour Java 1.x ... 3718
AWS Exemples de SDK pour Go 1.x ... 3722
Go et DAX ... 3722
AWS Exemples de SDK pour Node.js 2.x ... 3725
Node.js et DAX ... 3725
Historique de la documentation ... 3736
Mises à jour antérieures ... 3778

Fonctionnalités héritées 3817
Tables globales de la version 2017.11.29 (héritée) 3817
Comment ça marche 3818
Bonnes pratiques et exigences 3823
Création d'une table globale 3827
Surveillance des tables globales 3832
Utilisation d'IAM avec des tables globales 3834
Version précédente de l'API DynamoDB de bas niveau (2011-12-05) 3837
BatchGetItem 3838
BatchWriteItem 3846
CreateTable 3853
DeleteItem 3863
DeleteTable 3870
DescribeTables 3874
GetItem 3878
ListTables 3882
PutItem 3885
Requête 3893
Analyser 3911
UpdateItem 3932
UpdateTable 3942
Paramètres conditionnels hérités de DynamoDB 3947
AttributesToGet 3949
AttributeUpdates 3950
ConditionalOperator 3953
Expected 3954
KeyConditions 3960
QueryFilter 3963
ScanFilter 3965
Conditions d'écriture avec des paramètres hérités 3967
..... mmmcmlxxvii

Qu’est-ce qu’Amazon DynamoDB ?

Amazon DynamoDB est une base de données NoSQL distribuée, entièrement gérée et sans serveur offrant des performances à un chiffre en millisecondes, quelle que soit l'échelle.

DynamoDB vous permet de surmonter les complexités des bases de données relationnelles en matière d’opérations et de mise à l’échelle. DynamoDB est spécifiquement conçu et optimisé pour les charges de travail opérationnelles qui nécessitent des performances constantes à n’importe quelle échelle. Par exemple, DynamoDB fournit des performances constantes inférieures à 10 millisecondes pour les cas d’utilisation de panier d’achat, que vous ayez 10 ou 100 millions d’utilisateurs. Lancé en 2012, DynamoDB continue de vous aider à vous éloigner des bases de données relationnelles tout en réduisant les coûts et en améliorant les performances, quelle que soit l’échelle.

Des clients de toutes tailles, de tous secteurs et de toutes zones géographiques utilisent DynamoDB pour créer des applications modernes sans serveur qui peuvent commencer modestement et se mettre à l’échelle dans le monde entier. DynamoDB s’adapte pour prendre en charge des tables de pratiquement toutes tailles, tout en offrant des performances constantes inférieures à 10 millisecondes ainsi qu’une haute disponibilité.

Pour les événements tels que l’Amazon Prime Day, DynamoDB alimente plusieurs propriétés et systèmes Amazon à fort trafic, notamment Alexa, les sites Amazon.com et tous les centres de distribution Amazon. Pour de tels événements, DynamoDB a traité des APIs milliards d'appels provenant de propriétés et de systèmes Amazon. DynamoDB répond en permanence aux besoins de centaines de clients dont les tables doivent gérer des pics de trafic de plus d’un demi-million de demandes par seconde. Il répond également aux besoins de centaines de clients dont les tables font plus de 200 To et traitent plus d’un milliard de demandes par heure.

Rubriques

- Caractéristiques de DynamoDB
- Cas d’utilisation de DynamoDB
- Fonctionnalités de DynamoDB
- Intégrations de service
- Sécurité
- Résilience
- Accès à DynamoDB

- Tarification de DynamoDB
- Mise en route avec DynamoDB

Caractéristiques de DynamoDB

sans serveur

Avec DynamoDB, vous n'avez besoin de provisionner aucun serveur, ni de corriger, gérer, installer, maintenir ou exploiter aucun logiciel. DynamoDB assure la maintenance sans durée d'indisponibilité. Il n'implique aucune version (majeure, mineure ou correctif) ni aucune fenêtre de maintenance.

Le mode de capacité à la demande de DynamoDB propose des pay-as-you-go tarifs pour les demandes de lecture et d'écriture afin que vous ne payiez que pour ce que vous utilisez. Grâce au mode à la demande, DynamoDB augmente ou diminue verticalement et de manière instantanée la capacité de vos tables en fonction des besoins et préserve les performances sans aucune administration. Il réduit également verticalement la capacité à zéro afin que vous n'ayez pas de frais de débit à payer lorsque votre table n'a pas de trafic et qu'il n'y a pas de démarrage à froid.

NoSQL

En tant que base de données NoSQL, DynamoDB est spécifiquement conçu pour optimiser les performances, la capacité de mise à l'échelle, la gérabilité et la flexibilité par rapport aux bases de données relationnelles traditionnelles. Pour permettre un large éventail de cas d'utilisation, DynamoDB prend en charge à la fois les modèles clé-valeur et les modèles de données documentaires.

Contrairement aux bases de données relationnelles, DynamoDB ne prend pas en charge l'opérateur JOIN. Nous vous recommandons de dénormaliser votre modèle de données afin de réduire les allers-retours avec votre base de données et la puissance de traitement nécessaire pour répondre aux requêtes. En tant que base de données NoSQL, DynamoDB fournit une forte cohérence de lecture et des transactions ACID pour créer des applications de niveau entreprise.

Entièrement géré

En tant que service de base de données entièrement géré, DynamoDB prend en charge le gros du travail indifférencié lié à la gestion d'une base de données afin que vous puissiez vous concentrer sur la création de valeur pour vos clients. Il gère l'installation, les configurations, la maintenance, la haute disponibilité, la mise en service du matériel, la sécurité, les sauvegardes, la surveillance,

etc. Ainsi, lorsque vous créez une table DynamoDB, celle-ci est immédiatement prête pour les charges de travail de production. DynamoDB améliore constamment sa disponibilité, sa fiabilité, ses performances, sa sécurité et son fonctionnement sans nécessiter de mises à niveau ni de durée d'indisponibilité.

Performances inférieures à 10 millisecondes, quelle que soit l'échelle

DynamoDB est spécifiquement conçu pour améliorer les performances et la capacité de mise à l'échelle des bases de données relationnelles afin d'offrir des performances inférieures à 10 millisecondes à n'importe quelle échelle. Pour atteindre cette échelle et ces performances, DynamoDB est optimisé pour les charges de travail à hautes performances et APIs propose des solutions qui encouragent une utilisation efficace des bases de données. Il omet les fonctionnalités inefficaces et peu performantes à grande échelle, par exemple les opérations JOIN. DynamoDB garantit à votre application des performances constantes inférieures à 10 millisecondes, que vous comptiez 100 ou 100 millions d'utilisateurs.

Cas d'utilisation de DynamoDB

Des clients de toutes tailles, de tous secteurs et de toutes zones géographiques utilisent DynamoDB pour créer des applications modernes sans serveur qui peuvent commencer modestement et se mettre à l'échelle dans le monde entier. DynamoDB est idéal pour les cas d'utilisation qui nécessitent des performances constantes, quelle que soit l'échelle, avec des frais opérationnels minimes, voire nuls. La liste suivante présente certains cas d'utilisation adaptés à DynamoDB :

- Applications de services financiers : supposons que vous soyez une société de services financiers qui développe des applications, telles que le trading et le routage en temps réel, la gestion des prêts, la génération de jetons et les registres de transactions. Grâce aux tables globales DynamoDB, vos applications peuvent répondre aux événements et gérer le trafic de votre Régions AWS choix avec des performances de lecture et d'écriture rapides et locales.

  DynamoDB convient aux applications soumises aux exigences de disponibilité les plus strictes. Il élimine le fardeau opérationnel lié à la mise à l'échelle manuelle des instances pour faire face à la hausse du stockage ou du débit, à la gestion des versions et aux licences.

  Vous pouvez utiliser les transactions DynamoDB pour assurer l'atomicité, la cohérence, l'isolement et la durabilité (ACID) sur une ou plusieurs tables avec une seule requête. Les transactions (ACID) conviennent aux charges de travail qui incluent le traitement des transactions financières ou l'exécution des commandes. DynamoDB s'adapte instantanément à l'accroissement ou à la

baisse de vos charges de travail, ce qui vous permet de mettre à l'échelle votre base de données efficacement en fonction des conditions du marché, telles que les heures de trading.

- Applications de jeu : en tant que société de jeux, vous pouvez utiliser DynamoDB pour toutes les parties des plateformes de jeu, par exemple pour l'état du jeu, les données des joueurs, l'historique des sessions et les classements. Choisissez DynamoDB pour sa capacité de mise à l'échelle, ses performances constantes et la facilité de fonctionnement qu'offre son architecture sans serveur. DynamoDB convient parfaitement aux architectures évolutives nécessaires aux jeux les plus populaires. Il met rapidement à l'échelle le débit de votre jeu à la hausse ou à la baisse (avec la possibilité de réduire la capacité à zéro sans démarrage à froid). Cette capacité de mise à l'échelle optimise l'efficacité de votre architecture, que vous augmentiez la capacité horizontalement en cas de pic de trafic ou que vous la réduisiez lorsque l'utilisation du jeu est faible.
- Applications de streaming : les entreprises du secteur des médias et du divertissement utilisent DynamoDB comme index de métadonnées pour le contenu, le service de gestion de contenu ou la diffusion de statistiques sportives en temps quasi réel. Elles utilisent également DynamoDB pour exécuter des services utilisateur de liste de suivi et de mise en favoris et pour traiter des milliards d'événements de clients quotidiens afin de générer des recommandations. Ces clients bénéficient de la capacité de mise à l'échelle, des performances et de la résilience de DynamoDB. DynamoDB s'adapte à l'accroissement ou à la baisse des charges de travail, ce qui donne lieu à des cas d'utilisation de streaming multimédia pouvant répondre à tous les niveaux de la demande.

Pour en savoir plus sur la façon dont les clients de différents secteurs utilisent DynamoDB, consultez Amazon DynamoDB Customers et This is My Architecture.

Fonctionnalités de DynamoDB

Réplication multiactive avec les tables globales

Les tables globales fournissent une réplication multiactive de vos données sur l'ensemble de votre choix Régions AWS avec une disponibilité de 99,999 %. Les tables globales offrent une solution entièrement gérée permettant déployer une base de données multiactive et multirégions sans avoir à créer ni à gérer votre propre solution de réplication. Avec les tables globales, vous pouvez spécifier l' Régions AWS endroit où vous souhaitez que les tables soient disponibles. DynamoDB réplique les modifications de données en cours dans toutes ces tables.

Vos applications distribuées dans le monde entier peuvent accéder aux données localement dans les régions que vous avez sélectionnées pour atteindre des performances de lecture et d'écriture

inférieures à 10 millisecondes. Les tables globales étant multiactives, vous n'avez pas besoin d'une table principale. En d'autres termes, il n'y a pas de basculement compliqué ou différé, ni d'indisponibilité de la base de données en cas de basculement d'une application entre régions.

Transactions ACID

DynamoDB est conçu pour les charges de travail stratégiques. Il prend en charge les transactions (ACID) pour les applications qui nécessitent une logique métier complexe. DynamoDB fournit un support natif côté serveur pour les transactions, simplifiant ainsi l'expérience des développeurs lorsqu'il s'agit d'apporter des modifications coordonnées à plusieurs éléments dans et all-or-nothing entre les tables.

Capture des données modifiées pour les architectures basées sur les événements

DynamoDB prend en charge le streaming des registres de capture des données modifiées (CDC) au niveau des éléments en quasi-temps réel. Il offre deux modèles de streaming pour la capture des données modifiées : DynamoDB Streams et Kinesis Data Streams pour DynamoDB. Chaque fois qu'une application crée, met à jour ou supprime des éléments dans une table, Streams enregistre une séquence chronologique de chaque modification au niveau des éléments en temps quasi réel. DynamoDB Streams est donc la solution idéale pour les applications dotées d'une architecture axée sur les événements afin de prendre en compte les modifications et d'agir en conséquence.

Index secondaires

DynamoDB offre la possibilité de créer des index secondaires globaux et locaux, qui vous permettent d'interroger les données de la table à l'aide d'une clé alternative. Ces index secondaires vous permettent d'accéder aux données avec des attributs autres que la clé primaire, ce qui offre une flexibilité maximale pour l'accès à vos données.

Intégrations de service

DynamoDB s'intègre largement à Services AWS plusieurs d'entre eux pour vous aider à tirer le meilleur parti de vos données, à éliminer les charges lourdes indifférenciées et à gérer vos charges de travail à grande échelle. Voici quelques exemples : Amazon AWS CloudFormation CloudWatch, Amazon S3 Gestion des identités et des accès AWS (IAM) et AWS Auto Scaling. Les sections suivantes décrivent certaines des intégrations de service que vous pouvez exécuter à l'aide de DynamoDB :

Intégrations sans serveur

Pour créer des applications end-to-end sans serveur, DynamoDB s'intègre nativement à un certain nombre d'applications sans serveur. Services AWS Par exemple, vous pouvez intégrer DynamoDB à AWS Lambda pour créer des déclencheurs, qui sont des éléments de code répondant automatiquement aux événements dans DynamoDB Streams. Avec les déclencheurs, vous pouvez créer des applications basées sur les événements, qui réagissent aux modifications de données dans les tables DynamoDB. Pour optimiser les coûts, vous pouvez filtrer les événements traités par Lambda à partir d'un flux DynamoDB.

La liste suivante présente quelques exemples d'intégrations sans serveur avec DynamoDB :

- AWS AppSyncpour créer GraphQL APIs
- Amazon API Gateway pour créer REST APIs
- Lambda pour le calcul sans serveur
- Amazon Kinesis Data Streams pour la capture des données modifiées (CDC)

Importation et exportation de données vers Amazon S3

L'intégration de DynamoDB dans Amazon S3 vous permet d'exporter facilement des données vers un compartiment Amazon S3 à des fins d'analytique et de machine learning. DynamoDB prend en charge les exportations de tables complètes et les exportations incrémentielles pour exporter les données modifiées, mises à jour ou supprimées pendant une période de temps spécifiée. Vous pouvez également importer les données d'Amazon S3 dans une nouvelle table DynamoDB.

Intégration zéro ETL

DynamoDB prend en charge l'intégration zéro ETL avec Amazon Redshift et l'utilisation d'un pipeline d'ingestion OpenSearch avec Amazon DynamoDB. Ces intégrations vous permettent d'exécuter une analytique complexe et d'utiliser des fonctionnalités de recherche avancées au niveau des données de vos tables DynamoDB. Par exemple, vous pouvez effectuer une recherche vectorielle et en texte intégral, ainsi qu'une recherche sémantique sur vos données DynamoDB. Les intégrations zéro ETL n'ont aucun impact sur les charges de travail de production exécutées sur DynamoDB.

Mise en cache

DynamoDB Accelerator (DAX) est un service de mise en cache hautement disponible et entièrement géré conçu pour DynamoDB. DAX offre des performances jusqu'à 10 fois supérieures, de l'ordre de

quelques microsecondes au lieu de quelques millisecondes, même lorsque le nombre de requêtes s'élève à plusieurs millions par seconde. DAX prend en charge toutes les tâches nécessaires pour ajouter une accélération en mémoire à vos tables DynamoDB, sans que vous ayez à gérer l'invalidation du cache, le remplissage des données ni la gestion des clusters.

Sécurité

DynamoDB utilise IAM pour vous aider à contrôler en toute sécurité l'accès à vos ressources DynamoDB. Avec IAM, vous pouvez gérer de manière centralisée les autorisations qui déterminent quels utilisateurs DynamoDB peuvent accéder aux ressources. Vous pouvez utiliser IAM pour contrôler les personnes qui s'authentifient (sont connectées) et sont autorisées (disposent d'autorisations) à utiliser des ressources. Comme DynamoDB utilise IAM, aucun nom d'utilisateur ni mot de passe n'est nécessaire pour accéder à DynamoDB. Comme vous n'avez aucune politique complexe de rotation des mots de passe à gérer, la sécurité est simplifiée. Avec IAM, vous pouvez également activer un contrôle d'accès précis pour fournir une autorisation au niveau des attributs. Vous pouvez également définir des politiques basées sur les ressources avec la prise en charge de l'analyseur d'accès IAM et du blocage de l'accès public (BPA) afin de simplifier la gestion des politiques.

Par défaut, DynamoDB chiffre toutes les données client au repos. Le chiffrement au repos améliore la sécurité de vos données en utilisant les clés de chiffrement stockées dans AWS Key Management Service (AWS KMS). Le chiffrement au repos vous permet de créer des applications sensibles en matière de sécurité qui sont conformes aux exigences réglementaires et de chiffrement strictes. Lorsque vous accédez à une table chiffrée, DynamoDB déchiffre les données de la table de manière transparente. Vous n'avez pas besoin de changer de code ou d'application pour utiliser ou gérer des tables chiffrées. DynamoDB continue d'offrir la latence inférieure à 10 millisecondes habituelle, et toutes les requêtes DynamoDB fonctionnent sans heurt sur vos données chiffrées.

Vous pouvez spécifier si DynamoDB doit utiliser Clé détenue par AWS une (type de chiffrement par défaut) Clé gérée par AWS ou une clé gérée par le client pour chiffrer les données utilisateur. Le chiffrement par défaut à l'aide de clés KMS appartenant àAWS est disponible sans frais supplémentaires. Pour le chiffrement côté client, vous pouvez utiliser le kit SDK AWS Database Encryption.

DynamoDB respecte également plusieurs normes de conformité (HIPAA, PCI DSS et RGPD, par exemple), ce qui vous permet de répondre aux exigences réglementaires.

Résilience

Par défaut, DynamoDB réplique automatiquement vos données sur trois zones de disponibilité afin de garantir une durabilité élevée et un SLA de disponibilité de 99,99 %. DynamoDB fournit également des fonctionnalités supplémentaires pour vous aider à atteindre vos objectifs de continuité d'activité et de reprise après sinistre.

DynamoDB inclut les fonctionnalités suivantes qui contribuent à répondre à vos besoins en matière de résilience et de sauvegarde de données :

Fonctionnalités

- Tables globales
- Sauvegardes et point-in-time restauration continues
- Sauvegarde et restauration à la demande

Tables globales

Les tables globales DynamoDB garantissent un SLA de disponibilité de 99,999 % et une résilience multirégionale. Cela vous permet de créer des applications résilientes et de les optimiser pour un objectif de délai de reprise (RTO) et un objectif de point de reprise (RPO) aussi faibles que possible. Les tables globales s'intègrent également à AWS Fault Injection Service (AWS FIS) pour effectuer des expériences d'injection de pannes sur les charges de travail de vos tables globales. (Par exemple, suspendre la réplication globale d'une table vers n'importe quelle table répliquée.)

Sauvegardes et point-in-time restauration continues

Les sauvegardes continues vous offrent une granularité par seconde et la possibilité de lancer une point-in-time restauration. Grâce à point-in-time la restauration, vous pouvez restaurer une table à n'importe quel point dans le temps, jusqu'à la seconde au cours des 35 derniers jours. Vous pouvez définir la période de restauration sur une valeur comprise entre 1 et 35 jours.

Les sauvegardes continues et le lancement d'une point-in-time restauration n'utilisent pas la capacité allouée. Ils n'ont pas non plus d'impact sur les performances ni sur la disponibilité de vos applications.

Sauvegarde et restauration à la demande

Les fonctionnalités de sauvegarde et restauration à la demande vous permettent de créer des sauvegardes complètes d'une table pour l'archivage et la conservation à long terme, à des fins de conformité réglementaire. Les sauvegardes n'ont aucun impact sur les performances de la table, et vous pouvez sauvegarder des tables de n'importe quelle taille. Grâce à AWS Backup l'intégration, vous pouvez AWS Backup planifier, copier, étiqueter et gérer automatiquement le cycle de vie de vos sauvegardes à la demande DynamoDB. Vous pouvez ainsi copier des sauvegardes à la demande entre comptes et régions, et transférer les anciennes sauvegardes vers un stockage à froid pour optimiser les coûts. AWS Backup

Accès à DynamoDB

Vous pouvez utiliser DynamoDB en utilisant AWS Management Consolele, AWS Command Line Interfacele NoSQL Workbench pour DynamoDB ou DynamoDB. APIs

Pour de plus amples informations, veuillez consulter Accès à DynamoDB.

Tarification de DynamoDB

DynamoDB facture la lecture, l'écriture et le stockage de données dans vos tables, ainsi que toutes les fonctionnalités facultatives que vous choisissez d'activer. DynamoDB propose deux modes de capacité, qui offrent des options de facturation respectives pour traiter les lectures et les écritures dans vos tables : à la demande et provisionné.

DynamoDB est également inclus dans l'offre toujours gratuite, qui fournit 25 Go de stockage. L'offre toujours gratuite inclut également 25 unités de capacité d'écriture et 25 unités de capacité de lecture provisionnées (WCU, RCU), ce qui est suffisant pour traiter 200 millions de demandes par mois.

Pour plus d'informations, consultez Tarification Amazon DynamoDB.

Mise en route avec DynamoDB

Si vous utilisez DynamoDB pour la première fois, nous vous recommandons de commencer par lire les rubriques suivantes :

- Mise en route avec DynamoDB : vous guide tout au long du processus de configuration de DynamoDB, de création des exemples de tables et de chargement des données. Cette rubrique

fournit également des informations sur l'exécution de certaines opérations de base de données de base à l' AWS Management Console aide de NoSQL Workbench et DynamoDB. AWS CLI APIs

- Composants principaux de DynamoDB : décrit les concepts de base de DynamoDB.
- Bonnes pratiques de conception et d'architecture avec DynamoDB : fournit des recommandations concernant la conception NoSQL, DynamoDB Well-Architected Lens, la conception de tables et plusieurs autres fonctionnalités de DynamoDB. Ces bonnes pratiques vous aideront à optimiser les performances et à minimiser les coûts de débit lorsque vous travaillerez avec DynamoDB.

Nous vous recommandons également de consulter les didacticiels suivants qui présentent end-to-end des procédures complètes pour vous familiariser avec DynamoDB. Vous pouvez suivre ces didacticiels en utilisant la fonctionnalité de l'offre toujours gratuite.

- Créer et interroger une table NoSQL avec Amazon DynamoDB
- Créer une application à l'aide d'un magasin de données clé-valeur NoSQL

Pour plus d'informations sur les ressources, les outils et les stratégies de migration vers DynamoDB, consultez Migration vers DynamoDB. Pour lire les derniers blogs et livres blancs, consultez Amazon DynamoDB resources.

Mise en route avec DynamoDB

Vous apprendrez à vous connecter aux tables DynamoDB, à les créer et à les gérer dans les sections suivantes.

Avant de commencer, nous vous recommandons de vous familiariser avec les concepts de base d'Amazon DynamoDB. Vous pouvez obtenir un aperçu rapide dans Qu'est-ce qu'Amazon DynamoDB ? et un examen plus approfondi dans Composants de base d'Amazon DynamoDB. Passez ensuite aux prérequis.

> **Note**
>
> Lorsque vous vous inscrivez AWS, vous pouvez commencer à utiliser DynamoDB en utilisant AWS le niveau gratuit. Si vous n'avez pas encore profité de tous les avantages de l'offre gratuite pour Amazon DynamoDB, l'exécution des exemples de cette section ne vous coûtera rien. Sinon, vous devrez payer les frais d'utilisation DynamoDB standard à partir du moment où vous créez les tables et jusqu'à ce que vous les supprimiez.
> Si vous ne voulez pas vous inscrire à un compte gratuit, vous pouvez configurer DynamoDB local (version téléchargeable) sur votre ordinateur. La version téléchargeable vous permet de développer et de tester des applications localement sans créer de AWS compte ni accéder au service Web DynamoDB.

Rubriques

- Ressources Amazon DynamoDB pour les nouveaux utilisateurs
- Accès à DynamoDB
- Conditions préalables
- Configuration de DynamoDB
- Étape 1 : création d'une table dans DynamoDB
- Étape 2 : écrire des données dans une table DynamoDB
- Étape 3 : lire des données à partir d'une table DynamoDB
- Étape 4 : mettre à jour des données dans une table DynamoDB
- Étape 5 : interroger des données dans une table DynamoDB
- Étape 6 : (Facultatif) supprimer votre table DynamoDB pour nettoyer les ressources
- Poursuivre l'apprentissage concernant DynamoDB

- Générez du code d'infrastructure pour Amazon DynamoDB à l'aide de Console-to-Code

Ressources Amazon DynamoDB pour les nouveaux utilisateurs

Si vous êtes un nouvel utilisateur, nous vous conseillons de commencer par lire les sections suivantes et de vous y référer dès que vous en avez besoin :

- Service highlights and pricing – La page de détails des produits fournit une description générale de DynamoDB, des cas d'utilisation courants, une description des éléments principaux du service et la tarification qui s'applique.
- DynamoDB resources – La section DynamoDB resources contient des vidéos, des didacticiels et des conseils prescriptifs qui vous présentent le service, les concepts de modélisation des données et les fonctionnalités et fonctionnalités de base.
- Getting started – La section Mise en route avec DynamoDB contient des informations sur la configuration de DynamoDB, la création d'exemples de tables et le téléchargement de données. Elle fournit également des informations sur l'exécution des opérations de base de données clés à l'aide de la AWS Management Console, de l'AWS CLI, de NoSQL Workbench et des API DynamoDB.
- DynamoDB basics course – Cours numérique gratuit qui enseigne les principes de base de DynamoDB, notamment la conception de tables, les types de données et les opérations de base. Ce cours est idéal pour les développeurs qui découvrent les bases de données DynamoDB et NoSQL.
- DynamoDB Nuggets – Collection de didacticiels vidéo courts et ciblés qui expliquent les principaux concepts et fonctionnalités de DynamoDB. Ces petites vidéos abordent des sujets tels que la modélisation des données, les stratégies de partitionnement, les modèles d'accès et les meilleures pratiques dans un format facile à comprendre.
- DynamoDB code samples repository – Fournit des exemples pratiques de code DynamoDB dans différents langages de programmation.
- Free DynamoDB training – AWS Fournit des cours de formation numériques gratuits qui abordent les concepts, les fonctionnalités et les meilleures pratiques de DynamoDB. La série de cours approfondis DynamoDB vous aide à apprendre à concevoir des modèles de données efficaces, à implémenter des clés de partition efficaces et à utiliser efficacement les index secondaires.
- NoSQL Workbench for DynamoDB – NoSQL Workbench est un outil visuel unifié qui permet la modélisation des données, la visualisation des données et propose des fonctions de développement de requêtes. Il vous aide à concevoir, interroger et gérer des tables DynamoDB,

ce qui le rend particulièrement utile pour apprendre les concepts de modélisation de données de DynamoDB et tester des modèles d'accès.

- DynamoDB Design Patterns – Exemples de Bonnes pratiques et de modélisation de données illustrant les meilleures pratiques pour différents cas d'utilisation. Comprend des exemples pratiques et des exemples de code pour mettre en œuvre des solutions efficaces.
- Hands-on Tutorials – Didacticiels pas à pas dans la AWS Management Console qui vous guident dans les tâches courantes de DynamoDB, de la création de tables l'implémentation de requêtes complexes.
- Migrating to DynamoDB – Ce Migration vers DynamoDB fournit une vue d'ensemble du processus, des outils et des stratégies de migration d'une base de données vers DynamoDB.
- AWSWell-Architected Lens for DynamoDB – the section called "Cadre DynamoDB Well-Architected" fournit les bonnes pratiques architecturales en matière de conception et d'exploitation des applications fiables, sécurisées, efficaces et économiques à l'aide de DynamoDB. Il comprend des principes de conception, des modèles architecturaux et des directives opérationnelles alignés sur les six piliers du Well-Architected Framework.

Meilleures pratiques supplémentaires d'Amazon DynamoDB pour les nouveaux utilisateurs

Une fois que vous avez terminé les sections précédentes, lisez les sections suivantes :

- Capacité de débit DynamoDB

  Fournit une vue d'ensemble des deux modes de débit disponibles pour DynamoDB et explique comment sélectionner le mode de capacité approprié pour votre application. Le mode à la demande est l'option de débit par défaut et recommandée pour la plupart des charges de travail DynamoDB.
- Bonnes pratiques

  Identifiez et résolvez les problèmes afin d'optimiser les performances et de minimiser les coûts lorsque vous travaillez avec DynamoDB.

Ressources AWS CLI

Si vous souhaitez utiliser l'interface de ligne de commande AWS (AWS CLI), ces documents peuvent vous aider à la mise en route :

- Documentation AWS CLI

  Cette section fournit des informations sur le téléchargement et la mise en place de l'AWS CLI, la mise en place de l'AWS CLI dans votre système, ainsi que la définition de vos informations d'identification AWS.

- Documentation de l'AWS CLId pour DynamoDB

  Ceci est un document distinct qui couvre toutes les commandes AWS CLI pour DynamoDB, y compris leur syntaxe et des exemples.

Ressources de programmation

Vous pouvez développer des applications pour utiliser l'API DynamoDB avec différents langages de programmation courants. Voici quelques ressources :

- Outils pour Amazon Web Services

  AWS fournit plusieurs kits de développement (SDK) qui prennent en charge DynamoDB. Vous pouvez programmer pour DynamoDB en utilisant Java, .NET, PHP, Ruby, ainsi que d'autres langages. Ces kits de développement logiciel peuvent simplifier considérablement le développement de vos applications en formatant vos requêtes pour DynamoDB pour analyser syntaxiquement les réponses et fournir des logiques de nouvelle tentative et une gestion des erreurs.

- Référence de l'API DynamoDB

  Si vous ne voulez pas utiliser les kits SDK AWS, vous pouvez interagir avec DynamoDB directement en utilisant l'API DynamoDB. Ce document couvre toutes les opérations de l'API DynamoDB, y compris leur syntaxe et des exemples. Cette section fournit des astuces de dépannage et des informations pour créer et authentifier les requêtes et la façon de gérer les réponses.

Accès à DynamoDB

Vous pouvez accéder à Amazon DynamoDB à l'aide de l'API DynamoDB, AWS Management Console de la AWS CLI() ou AWS Command Line Interface de l'API DynamoDB.

Rubriques

- Utilisation de la console
- À l'aide du AWS CLI
- Utilisation de l’API
- Utilisation de NoSQL Workbench pour DynamoDB
- Plages d’adresses IP
- Points de terminaison à double pile pour le protocole Internet version 6 () IPv6

Utilisation de la console

Vous pouvez accéder au AWS Management Console pour Amazon https://console.aws.amazon.com/dynamodb/ DynamoDB depuis votre domicile.

Voici certaines des actions que vous pouvez effectuer dans la console DynamoDB :

- Gestion des tables : créez, mettez à jour et supprimez des tables. Le calculateur de capacité peut vous aider à estimer vos besoins en capacité.
- Interaction avec les données : consultez, ajoutez, mettez à jour et supprimez des éléments dans vos tables. Gérez les paramètres de durée de vie (TTL).
- Surveillance et analyse : consultez les tableaux de bord, surveillez et configurez les alarmes et analysez les métriques et les alertes pour vos tables DynamoDB.
- Optimisation et extension : gérez les index secondaires, les flux, les déclencheurs, la capacité réservée et les autres fonctionnalités avancées pour améliorer votre utilisation de DynamoDB.

La console DynamoDB fournit une interface complète vous permettant de gérer vos ressources DynamoDB. Nous vous encourageons à accéder à la console et à interagir avec elle pour en savoir plus.

À l'aide du AWS CLI

Vous pouvez utiliser le AWS Command Line Interface (AWS CLI) pour contrôler plusieurs AWS services à partir de la ligne de commande et les automatiser par le biais de scripts. Vous pouvez utiliser le AWS CLI pour des opérations ad hoc, telles que la création d'une table. Vous pouvez également l’utiliser pour incorporer des opérations Amazon DynamoDB au sein de scripts utilitaires.

Avant de pouvoir utiliser le AWS CLI avec DynamoDB, vous devez obtenir un ID de clé d'accès et une clé d'accès secrète. Pour de plus amples informations, veuillez consulter Accorder un accès par programmation .

Pour obtenir la liste complète de toutes les commandes disponibles pour DynamoDB dans AWS CLI le, consultez AWS CLI la référence des commandes.

Téléchargement et configuration du AWS CLI

AWS CLI Il est disponible à l'adresse http://aws.amazon.com/cli. Elle s'exécute sous Windows, macOS ou Linux. Après avoir téléchargé le AWS CLI, procédez comme suit pour l'installer et le configurer :

1. Accédez au Guide de l'utilisateur AWS Command Line Interface.
2. Suivez les instructions d'Installation de l' AWS CLI et de Configuration de l' AWS CLI.

Utilisation du AWS CLI avec DynamoDB

Le format de ligne de commande se compose d'un nom d'opération DynamoDB, suivi des paramètres pour cette opération. Il AWS CLI prend en charge une syntaxe abrégée pour les valeurs des paramètres, ainsi que du JSON.

Par exemple, la commande suivante permet de créer une table nommée Music. La clé de partition est Artist et la clé de tri est SongTitle. (Pour une lecture plus facile, les commandes longues dans cette section sont divisées en plusieurs lignes).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Les commandes suivantes permettent d'ajouter de nouveaux éléments à la table. Ces exemples utilisent une combinaison de la syntaxe raccourcie et de JSON.

```
aws dynamodb put-item \
```

```
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL

aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

Sur la ligne de commande, il peut être difficile de composer un JSON valide. Toutefois, l' AWS CLI peut lire les fichiers JSON. Par exemple, imaginons l'extrait de code JSON suivant qui est stocké dans un fichier nommé key-conditions.json.

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

Vous pouvez désormais émettre une demande de `Query` à l'aide de l' AWS CLI. Dans cet exemple, le contenu du fichier key-conditions.json est utilisé pour le paramètre `--key-conditions`.

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

Utilisation du AWS CLI avec DynamoDB local

Ils AWS CLI peuvent également interagir avec DynamoDB local (version téléchargeable) qui s'exécute sur votre ordinateur. Pour ce faire, ajoutez le paramètre suivant pour chaque commande :

`--endpoint-url http://localhost:8000`

L'exemple suivant utilise le AWS CLI pour répertorier les tables d'une base de données locale.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Si DynamoDB utilise un numéro de port autre que la valeur par défaut (8000), modifiez la valeur `--endpoint-url` en conséquence.

> **Note**
>
> Impossible AWS CLI d'utiliser le DynamoDB local (version téléchargeable) comme point de terminaison par défaut. Par conséquent, vous devez spécifier `--endpoint-url` avec chaque commande.

Utilisation de l'API

Vous pouvez utiliser le AWS Management Console et le AWS Command Line Interface pour travailler de manière interactive avec Amazon DynamoDB. Toutefois, pour tirer le meilleur parti de DynamoDB, vous pouvez écrire du code d'application à l'aide du. AWS SDKs

Ils AWS SDKs fournissent une prise en charge étendue de DynamoDB en Java JavaScript , dans le navigateur, .NET, Node.js,PHP, Python, Ruby, C++, Go , Android et iOS.

Avant de pouvoir utiliser le AWS SDKs avec DynamoDB, vous devez obtenir AWS un ID de clé d'accès et une clé d'accès secrète. Pour de plus amples informations, veuillez consulter Configuration de DynamoDB (service web) .

Pour une présentation détaillée de la programmation d'applications DynamoDB à l'aide AWS SDKs du, voir. Programmation avec DynamoDB et AWS SDKs

Utilisation de NoSQL Workbench pour DynamoDB

Vous pouvez également accéder à DynamoDB en téléchargeant et en utilisant le NoSQL Workbench pour DynamoDB.

NoSQL Workbench pour Amazon DynamoDB est une application côté client multiplateforme pour le développement et les opérations de base de données moderne. Il est disponible pour Windows, macOS et Linux. NoSQL Workbench est un outil de développement visuel qui propose des fonctionnalités de modélisation de données, de visualisation de données et de développement de requêtes, afin de vous aider à concevoir, créer, interroger et gérer des tables DynamoDB. NoSQL Workbench inclut désormais DynamoDB local en tant que composant facultatif du processus d'installation, ce qui facilite la modélisation des données dans DynamoDB local. Pour en savoir plus sur DynamoDB local et ses exigences, consultez Configuration de DynamoDB Local (version téléchargeable) .

> **Note**
>
> Le NoSQL Workbench pour DynamoDB ne prend actuellement pas en charge les AWS connexions configurées avec l'authentification à deux facteurs (2FA).

Modélisation des données

NoSQL Workbench pour DynamoDB vous permet de créer des modèles de données ou de concevoir des modèles à partir de modèles de données existants, qui pourront satisfaire les modèles d'accès aux données de votre application. À la fin du processus, vous pouvez également importer et exporter le modèle de données conçu. Pour plus d'informations, consultez Création de modèles de données avec NoSQL Workbench.

Visualisation des données

Le visualiseur du modèle de données fournit un canevas sur lequel vous pouvez mapper des requêtes et visualiser les modèles d'accès (facettes) de l'application sans avoir à écrire de code. Chaque facette correspond à un modèle d'accès différent dans DynamoDB. Vous pouvez générer automatiquement des données d'exemple à utiliser dans votre modèle de données. Pour plus d'informations, consultez Visualisation des modèles d'accès de données.

Création des opérations

NoSQL Workbench fournit une interface utilisateur graphique enrichie pour développer et tester des requêtes. Vous pouvez utiliser le créateur d'opérations pour consulter, explorer et interroger des ensembles de données en temps réel. Vous pouvez également utiliser le créateur d'opérations structurées pour créer et réaliser des opérations de plan de données. Il prend en charge les expressions de projection et de condition, et vous permet de générer un exemple de

code dans plusieurs langages. Pour de plus amples informations, veuillez consulter Exploration des jeux de données et des opérations de création avec NoSQL Workbench.

Plages d'adresses IP

Amazon Web Services (AWS) publie ses plages d'adresses IP actuelles au format JSON. Pour afficher les plages actuelles, téléchargez ip-ranges.json. Pour plus d'informations, consultez AWS IP Address Ranges dans le manuel Références générales AWS.

Pour trouver les plages d'adresses IP que vous utilisez pour accéder aux tables et index DynamoDB, recherchez la chaîne suivante dans le fichier ip-ranges.json : `"service": "DYNAMODB"`.

> **Note**
>
> Les plages d'adresses IP ne s'appliquent pas aux flux DynamoDB ni à DynamoDB Accelerator (DAX).

Points de terminaison à double pile pour le protocole Internet version 6 () IPv6

DynamoDB propose des points de terminaison à double pile compatibles avec et. IPv4 IPv6 Les conventions de dénomination des points de terminaison sont les suivantes :

- `dynamodb.<region>.api.aws`
- `<account-id>.ddb.<region>.api.aws`
- `streams-dynamodb.<region>.api.aws`
- `dax.<region>.api.aws`
- `dynamodb-fips.<region>.api.aws`

Pour obtenir la liste complète des points de terminaison DynamoDB et de leur disponibilité régionale, consultez la rubrique Points de terminaison et quotas Amazon DynamoDB dans le guide de référence général.AWS

Pour plus d'informations sur la configuration de la AWS CLI pour utiliser des points de terminaison à double pile, voir la section Configurer pour utiliser des points de terminaison à double pile pour tous les AWS services du guide de l'interface de AWS ligne de commande.

Pour plus d'informations sur la configuration de vos clients SDK pour utiliser des points de terminaison à double pile, consultez la rubrique Points de terminaison à double pile et FIPS dans le guide des outils et.AWS SDKs

Avant d'utiliser DynamoDB IPv6 avec, vous devez mettre à jour votre rôle d'utilisateur IAM ou les politiques basées sur les ressources que vous utilisez pour le filtrage des adresses IP afin d'inclure les plages d'adresses. IPv6 Les politiques de filtrage des adresses IP qui ne tiennent pas compte de IPv6 l'adresse peuvent entraîner des problèmes d'accès. Pour plus d'informations, consultez la section Opérateurs de condition des adresses IP du guide AWS Identity and Access Management.

Conditions préalables

Avant de démarrer le didacticiel Amazon DynamoDB, découvrez comment accéder à DynamoDB dans Accès à DynamoDB. Configurez ensuite DynamoDB à l'aide du service Web ou de la version téléchargée localement dans Configuration de DynamoDB . Après cela, passez à Étape 1 : création d'une table dans DynamoDB.

> **Note**
>
> - Si vous prévoyez d'interagir avec DynamoDB uniquement via AWS Management Console le, vous n'avez pas besoin AWS de clé d'accès. Suivez les étapes de la section S'inscrire àAWS, puis passez àÉtape 1 : création d'une table dans DynamoDB.
> - Si vous ne voulez pas vous inscrire à un compte gratuit, vous pouvez configurer DynamoDB local (version téléchargeable). Passez ensuite à l'Étape 1 : création d'une table dans DynamoDB.
> - Il existe des différences lorsque vous utilisez des commandes CLI dans des terminaux sous Linux et Windows. Le guide suivant présente des commandes formatées pour les terminaux Linux (y compris macOS) et des commandes formatées pour Windows CMD. Choisissez la commande qui convient le mieux à l'application de terminal que vous utilisez.

Configuration de DynamoDB

Outre le service Web Amazon DynamoDB AWS, il fournit une version téléchargeable de DynamoDB que vous pouvez exécuter sur votre ordinateur. Elle s'avère utile pour développer et tester du code. Elle permet d'écrire et de tester des applications localement sans accéder au service web DynamoDB.

Les rubriques de cette section décrivent comment configurer DynamoDB (version téléchargeable) et le service web DynamoDB.

Rubriques

- Configuration de DynamoDB (service web)
- Configuration de DynamoDB Local (version téléchargeable)

Configuration de DynamoDB (service web)

Pour utiliser le service web Amazon DynamoDB :

1. Inscrivez-vous àAWS.
2. Obtenez une clé d'AWS accès (utilisée pour accéder à DynamoDB par programmation).

> **Note**
>
> Si vous prévoyez d'interagir avec DynamoDB uniquement via AWS Management Console le, vous n'avez pas besoin de cléAWS d'accès et vous pouvez passer directement à. Utilisation de la console

3. Configurez vos informations d'identification (pour accéder à DynamoDB par programme).

S'inscrire àAWS

Pour utiliser le service DynamoDB, vous devez disposer d'un compte.AWS Si vous n'en avez pas déjà un, vous serez invité à en créer un au moment de l'inscription. Les AWS services auxquels vous vous inscrivez ne vous sont pas facturés, sauf si vous les utilisez.

Pour vous inscrire àAWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources

de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

Accorder un accès par programmation

Avant de pouvoir accéder à DynamoDB par programmation ou via AWS Command Line Interface le AWS CLI(), vous devez disposer d'un accès par programmation. Vous n'avez pas besoin d'accès par programmation si vous prévoyez d'utiliser uniquement la console DynamoDB.

Les utilisateurs ont besoin d'un accès programmatique s'ils souhaitent interagir avec AWS l'extérieur du AWS Management Console. La manière d'accorder un accès programmatique dépend du type d'utilisateur qui y accède AWS.

Pour accorder aux utilisateurs un accès programmatique, choisissez l'une des options suivantes.

Quel utilisateur a besoin d'un accès programmatique ?	À	Méthode
IAM	(Recommandé) Utilisez les informations d'identification de la console comme informations d'identification temporaires pour signer les demandes programmatiques adressées au AWS CLIAWS SDKs, ou AWS APIs.	Suivez les instructions de l'interface que vous souhaitez utiliser. • Pour le AWS CLI, voir Connexion pour le développement AWS local dans le guide de AWS Command Line Interface l'utilisateur. • Pour AWS SDKs, voir Connexion pour le développement AWS local dans le guide de référence AWS SDKs and Tools.
Identité de la main-d'œuvre	Utilisez des informations d'identification temporaires pour signer les demandes	Suivez les instructions de l'interface que vous souhaitez utiliser.

Quel utilisateur a besoin d'un accès programmatique ?	À	Méthode
(Utilisateurs gérés dans IAM Identity Center)	programmatiques adressées au AWS CLIAWS SDKs, ou AWS APIs.	• Pour le AWS CLI, voir Configuration du AWS CLIà utiliser AWS IAM Identity Center dans le guide de AWS Command Line Interface l'utilisateur. • Pour AWS SDKs, outils, et AWS APIs, voir Authentification IAM Identity Center dans le guide de référence AWS SDKs et Tools.
IAM	Utilisez des informations d'identification temporaires pour signer les demandes programmatiques adressées au AWS CLIAWS SDKs, ou AWS APIs.	Suivez les instructions de la section Utilisation d'informations d'identification temporaires avec AWS les ressources du Guide de l'utilisateur IAM.

Quel utilisateur a besoin d'un accès programmatique ?	À	Méthode
IAM	(Non recommandé) Utilisez des informations d'identification à long terme pour signer des demandes programmatiques adressées au AWS CLIAWS SDKs, ou AWS APIs.	Suivez les instructions de l'interface que vous souhaitez utiliser. • Pour le AWS CLI, voir Authentification à l'aide des informations d'identification utilisateur IAM dans le Guide de l'AWS Command Line Interface utilisateur. • Pour les outils AWS SDKs et, voir Authentifier à l'aide d'informations d'identif ication à long terme dans le guide de référence des outils AWS SDKs et. • Pour AWS APIs, voir Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Configuration de vos informations d'identification

Avant de pouvoir accéder à DynamoDB par programmation ou via AWS CLI le, vous devez configurer vos informations d'identification afin d'autoriser vos applications.

Il existe plusieurs méthodes pour le faire. Par exemple, vous pouvez créer manuellement le fichier d'informations d'identification pour stocker votre ID de clé d'accès et votre clé d'accès secrète. Vous pouvez également utiliser la AWS CLI commande `aws configure` pour créer automatiquement le fichier. Vous pouvez également utiliser les variables d'environnement. Pour plus d'informations sur la configuration de vos informations d'identification, consultez le guide du développeur du AWS SDK spécifique à la programmation.

Pour installer et configurer le AWS CLI, voirÀ l'aide du AWS CLI.

Intégration à d’autres services DynamoDB

Vous pouvez intégrer DynamoDB à de nombreux autres services.AWS Pour plus d’informations, consultez les ressources suivantes :

- Utilisation de DynamoDB avec d'autres services AWS
- CloudFormation pour DynamoDB
- Utilisation de AWS Backup avec DynamoDB
- Gestion des identités et des accès AWS(IAM) et DynamoDB
- Utilisation d’AWS Lambda avec Amazon DynamoDB

Configuration de DynamoDB Local (version téléchargeable)

La version téléchargeable d’Amazon DynamoDB vous permet d’écrire et de tester des applications sans accéder au service web DynamoDB. Au lieu de cela, la base de données est autonome sur votre ordinateur. Lorsque vous êtes prêt à déployer votre application en production, vous supprimez le point de terminaison local dans le code afin que l’application pointe vers le service web DynamoDB.

Disposer de cette version locale vous aide à économiser sur les frais de transfert de données, de stockage de données et de débit. En outre, vous n’avez pas besoin d’une connexion internet pendant que vous développez votre application.

DynamoDB local est disponible en téléchargement (nécessite JRE) sous la forme d’une dépendance Apache Maven ou d’une image Docker.

Si vous préférez utiliser le service web Amazon DynamoDB, consultez Configuration de DynamoDB (service web) .

Rubriques

- Déploiement de DynamoDB localement sur votre ordinateur
- Notes d’utilisation de DynamoDB Local
- Historique des versions de DynamoDB local
- Télémétrie dans DynamoDB Local

Déploiement de DynamoDB localement sur votre ordinateur

> **Note**
>
> - DynamoDB local est disponible en trois versions : v3.x (actuelle), v2.x (héritée) et v1.x (obsolète).
> - DynamoDB v3.x est recommandé pour vos tests et développements locaux.
> - La migration de DynamoDB local V2.x vers V3.x nécessite la mise à jour des instructions d'importation de `com.amazonaws.services.dynamodbv2` vers `software.amazon.dynamodb` et la mise à jour des dépendances Maven pour les utilisateurs de Maven.
> - Si vous migrez une application qui utilise le kit SDK pour Java v1.x vers le kit SDK pour Java 2.x, suivez les étapes du kit AWS SDK pour Java 2.x.

Téléchargement de DynamoDB local

Suivez ces étapes pour configurer et exécuter DynamoDB sur votre ordinateur :

Pour configurer DynamoDB sur votre ordinateur

1. Téléchargez DynamoDB local gratuitement à partir de l'un des emplacements suivants.

Télécharger les liens	Sommes de chèques
.tar.gz | .zip	.tar.gz.sha256|.zip.sha256

> **Important**
>
> Pour exécuter DynamoDB v2.6.0 ou supérieures sur votre ordinateur, vous devez disposer de l'environnement d'exécution Java (JRE) version 17.x ou plus récente. L'application ne fonctionne pas sur les versions antérieures de JRE.

2. Une fois que vous avez téléchargé l'archive, extrayez le contenu et copiez le répertoire extrait vers un emplacement de votre choix.

3. Pour démarrer DynamoDB sur votre ordinateur, ouvrez une fenêtre d'invite de commande, accédez au répertoire où vous avez extrait `DynamoDBLocal.jar`, puis entrez la commande suivante.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> **Note**
>
> Si vous utilisez Windows PowerShell, veillez à inclure le nom du paramètre ou le nom complet et la valeur comme suit :
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> DynamoDB traite les demandes entrantes jusqu'à ce que vous l'arrêtiez. Pour arrêter DynamoDB, tapez Ctrl + C à l'invite de commande.
> DynamoDB utilise le port 8000 par défaut. Si le port 8000 n'est pas disponible, cette commande lève une exception. Pour obtenir la liste complète des options d'exécution de DynamoDB, y compris `-port`, entrez la commande suivante.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Pour pouvoir accéder à DynamoDB par programme ou via l'AWS Command Line Interface(AWS CLI), vous devez configurer vos informations d'identification pour permettre l'autorisation de vos applications. La version téléchargeable de DynamoDB requiert des informations d'identification pour fonctionner, comme l'illustre l'exemple suivant.

```
AWS Access Key ID: "fakeMyKeyId"AWS Secret Access Key: "fakeSecretAccessKey"
Default Region Name: "fakeRegion"
```

   Vous pouvez utiliser la commande `aws configure` de AWS CLI pour configurer des informations d'identification. Pour de plus amples informations, veuillez consulter À l'aide du AWS CLI.

5. Vous pouvez commencer à écrire des applications. Pour accéder à DynamoDB s'exécutant localement avec AWS CLI le, utilisez le paramètre. `--endpoint-url` Par exemple, utilisez la commande suivante pour afficher la liste des tables DynamoDB.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Exécution de DynamoDB local en tant qu'image Docker

La version téléchargeable d'Amazon DynamoDB est disponible sous la forme d'une image Docker. Pour plus d'informations, consultez dynamodb-local. Pour savoir quelle est votre version dans DynamoDB local, saisissez la commande suivante :

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

Pour un exemple d'utilisation de DynamoDB local dans le cadre d'une application REST basée sur AWS SAM(), consultez AWS Serverless Application Model l'application SAM DynamoDB pour la gestion des commandes. Cet exemple d'application montre comment utiliser DynamoDB Local à des fins de test.

Si vous souhaitez exécuter une application à conteneurs multiples qui utilise également le conteneur local DynamoDB, utilisez Docker Compose pour définir et exécuter tous les services de votre application, y compris DynamoDB local.

Pour installer et exécuter DynamoDB Local avec Docker Compose :

1. Téléchargez et installez Docker Desktop.
2. Copiez le code suivant dans un fichier et enregistrez-le sous le nom `docker-compose.yml`.

   ```
   services:
    dynamodb-local:
      command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
      image: "amazon/dynamodb-local:latest"
      container_name: dynamodb-local
      ports:
        - "8000:8000"
      volumes:
        - "./docker/dynamodb:/home/dynamodblocal/data"
      working_dir: /home/dynamodblocal
   ```

   Si vous voulez que votre application et DynamoDB Local soient dans des conteneurs distincts, utilisez le fichier yaml suivant :

   ```
   version: '3.8'
   services:
    dynamodb-local:
   ```

```
    command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
    image: "amazon/dynamodb-local:latest"
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
  app-node:
    depends_on:
      - dynamodb-local
    image: amazon/aws-cli
    container_name: app-node
    ports:
     - "8080:8080"
    environment:
      AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
      AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
    command:
      dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
  us-west-2
```

Ce script docker-compose.yml crée un conteneur `app-node` et un conteneur `dynamodb-local`. Le script exécute une commande dans le conteneur `app-node`, qui utilise l'AWS CLI pour se connecter au conteneur `dynamodb-local`, et décrit les limites de compte et de table.

Pour l'utiliser avec votre propre image d'application, remplacez la valeur `image` dans l'exemple ci-dessous par celle de votre application.

```
version: '3.8'
services:
  dynamodb-local:
    command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
    image: "amazon/dynamodb-local:latest"
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
  app-node:
    image: location-of-your-dynamodb-demo-app:latest
```

```
  container_name: app-node
  ports:
    - "8080:8080"
  depends_on:
    - "dynamodb-local"
  links:
    - "dynamodb-local"
  environment:
    AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
    AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
    REGION: 'eu-west-1'
```

> **Note**
>
> Les scripts YAML nécessitent que vous spécifiiez une clé d'AWS accès et une cléAWS secrète, mais il n'est pas nécessaire qu'il s'agisse de AWS clés valides pour accéder à DynamoDB local.

3. Exécutez la commande de ligne de commande suivante :

```
docker-compose up
```

Exécution de DynamoDB local en tant que dépendance Apache Maven

> **Note**
>
> Si vous migrez une application qui utilise le kit SDK pour Java v1.x vers le kit SDK pour Java 2.x, suivez les étapes du kit AWS SDK pour Java 2.x.

Suivez ces étapes pour utiliser Amazon DynamoDB dans votre application en tant que dépendance.

Pour déployer DynamoDB en tant que référentiel Apache Maven

1. Téléchargez et installez Apache Maven. Pour plus d'informations, consultez Downloading Apache Maven et Installing Apache Maven.
2. Ajoutez le référentiel Maven DynamoDB au fichier POM (Project Object Model) de votre application.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.1.0</version>
   </dependency>
</dependencies>
```

Exemple de modèle à utiliser avec Spring Boot 3 and/or Spring Framework 6 :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
   <spring-boot.version>3.0.1</spring-boot.version>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.0.1</version>
   </parent>

<dependencies>
   <dependency>
       <groupId>com.amazonaws</groupId>
       <artifactId>DynamoDBLocal</artifactId>
       <version>3.1.0</version>
   </dependency>
   <!-- Spring Boot -->
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Other Spring dependencies -->
    <!-- Replace the version numbers with the desired version -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>6.0.0</version>
    </dependency>
    <!-- Add other Spring dependencies as needed -->
    <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> **Note**
>
> Vous pouvez également utiliser l'URL du référentiel central Maven.

Exécutez DynamoDB en local dans AWS CloudShell

AWS CloudShell est un shell pré-authentifié basé sur un navigateur que vous pouvez lancer directement depuis le.AWS Management Console Vous pouvez naviguer AWS CloudShellAWS Management Console de différentes manières. Pour plus d'informations, consultez la section Mise en route avec AWS CloudShell.

Procédez comme suit pour exécuter DynamoDB en local n'importe où dans le AWS CloudShell.AWS Management Console

Pour exécuter DynamoDB en local dans votre AWS CloudShellAWS Management Console

1. Lancez AWS CloudShell depuis l'interface de la console Région AWS, choisissez un shell disponible et passez à votre shell préféré, tel que Bash ou Z. PowerShell
2. Pour en choisir une Région AWS, allez dans le menu Sélectionnez une région et sélectionnez une région prise en charge Région AWS. (Les régions disponibles sont surlignées.)
3. À partir du AWS Management Console, lancez-le AWS CloudShell en choisissant l'une des options suivantes :

   a. Dans la barre de navigation, choisissez l’icône AWS CloudShell.

   b. Dans la zone de recherche, entrez le mot CloudShell, puis choisissez CloudShell.

   c. Dans le widget Visites récentes, sélectionnez CloudShell.

   d. Dans la barre d'outils de la console, choisissez CloudShell.
4. Pour exécuter DynamoDB en local, vous pouvez AWS CloudShell utiliser l'alias. `dynamodb-local` Vous pouvez spécifier des options de ligne de commande supplémentaires pour modifier les paramètres DynamoDB local. Consultez the section called “Notes d’utilisation” pour connaître les options disponibles.

   > **Note**
   >
   > Pour exécuter DynamoDB local en arrière-plan, exécutez DynamoDB local en utilisant :.AWS CloudShell`dynamodb-local &`
5. Pour accéder à DynamoDB s'exécutant localement AWS CloudShell avec AWS CLI le, utilisez le paramètre. `--endpoint-url` Par exemple, utilisez la commande suivante pour afficher la liste des tables DynamoDB :

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

Pour un exemple de projet présentant plusieurs approches pour configurer et utiliser DynamoDB local, notamment le téléchargement de fichiers JAR, leur exécution sous forme d'image Docker et leur utilisation en tant que dépendance Maven, consultez DynamoDB Local Sample Java Project.

Notes d'utilisation de DynamoDB Local

À l'exception du point de terminaison, les applications qui s'exécutent avec la version téléchargeable d'Amazon DynamoDB doivent également utiliser le service web DynamoDB. Cependant, lorsque vous utilisez DynamoDB localement, vous devez être conscient de ce qui suit :

- Si vous utilisez `-sharedDb` cette option, DynamoDB crée un seul fichier de base de données nommé .db. shared-local-instance Tous les programmes qui se connectent à DynamoDB accèdent à ce fichier. Si vous supprimez le fichier, vous perdez toutes les données que vous y avez stockées.
- Si vous omettez`-sharedDb`, le fichier de base de données s'appelle myaccesskeyid_region.db, avec l'ID de cléAWS d'accès et AWS la région tels qu'ils apparaissent dans la configuration de votre application. Si vous supprimez le fichier, vous perdez toutes les données que vous y avez stockées.
- Si vous utilisez l'option `-inMemory`, DynamoDB n'écrit aucun fichier de base de données. Au lieu de cela, toutes les données sont écrites en mémoire et ne sont pas enregistrées lorsque vous arrêtez DynamoDB.
- Si vous utilisez l'option `-inMemory`, l'option `-sharedDb` est également nécessaire.
- Si vous utilisez l'option `-optimizeDbBeforeStartup`, vous devez également spécifier le paramètre `-dbPath` de sorte que DynamoDB puisse trouver son fichier de base de données.
- AWS SDKs Pour DynamoDB, la configuration de votre application nécessite que la configuration de votre application spécifie une valeur de clé d'accès et AWS une valeur de région. DynamoDB utilise ces valeurs pour nommer le fichier de base de données local, sauf si vous utilisez l'option `-sharedDb` ou `-inMemory`. Il n'est pas nécessaire que ces AWS valeurs soient valides pour être exécutées localement. Vous pouvez trouver pratique de choisir des valeurs valides afin de pouvoir exécuter votre code dans le cloud en modifiant simplement le point de terminaison que vous utilisez.
- DynamoDB local renvoie toujours une valeur nulle pour `billingModeSummary.`
- La variable d'environnement `AWS_ACCESS_KEY_ID` de DynamoDB local ne peut contenir que des lettres (A à Z, a à z) et des chiffres (0 à 9).
- DynamoDB local ne prend pas en Point-in-time charge la restauration (PITR).

Rubriques

- Options de ligne de commande
- Définition du point de terminaison local
- Différences entre la version téléchargeable de DynamoDB et le service web DynamoDB

Options de ligne de commande

Avec la version téléchargeable de DynamoDB, vous pouvez utiliser les options de ligne de commande suivantes :

- `-corsvalue`— Permet la prise en charge du partage de ressources entre origines (CORS) pour. JavaScript Vous devez fournir une liste « allow » séparée par des virgules de domaines spécifiques. Le paramètre par défaut de `-cors` est un astérisque (*), qui autorise l'accès public.
- `-dbPath value` – Répertoire dans lequel DynamoDB écrit son fichier de base de données. Si vous ne spécifiez pas cette option, le fichier est écrit dans le répertoire actuel. Vous ne pouvez pas spécifier à la fois `-dbPath` et `-inMemory`.
- `-delayTransientStatuses` – Amène DynamoDB à introduire des retards pour certaines opérations. DynamoDB (version téléchargeable) peut effectuer certaines tâches presque instantanément, create/update/delete telles que les opérations sur les tables et les index. Cependant, le service DynamoDB nécessite plus de temps pour exécuter ces tâches. La définition de ce paramètre permet à DynamoDB s'exécutant sur votre ordinateur de simuler plus fidèlement le comportement du service web DynamoDB. (Actuellement, ce paramètre provoque des retards uniquement pour les index secondaires globaux dont l'état est CREATING ou DELETING.)
- `-help` – Affiche un récapitulatif de l'utilisation et des options.
- `-inMemory` – DynamoDB s'exécute en mémoire au lieu d'utiliser un fichier de base de données. Lorsque vous arrêtez DynamoDB, aucune des données n'est enregistrée. Vous ne pouvez pas spécifier à la fois `-dbPath` et `-inMemory`.
- `-optimizeDbBeforeStartup` – Optimise les tables de base de données sous-jacentes avant de démarrer DynamoDB sur votre ordinateur. Vous devez également spécifier `-dbPath` lorsque vous utilisez ce paramètre.
- `-port value` – Numéro de port que DynamoDB utilise pour communiquer avec votre application. Si vous ne spécifiez pas cette option, le port par défaut est `8000`.

> **Note**
>
> DynamoDB utilise le port 8000 par défaut. Si le port 8000 n'est pas disponible, cette commande lève une exception. Vous pouvez utiliser l'option `-port` pour spécifier un autre numéro de port. Pour obtenir la liste complète des options d'exécution de DynamoDB, dont `-port`, entrez la commande suivante :
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb` – Si vous spécifiez `-sharedDb`, DynamoDB utilise un fichier de base de données unique au lieu de fichiers distincts pour chaque identification et région.
- `-disableTelemetry` : lorsque cette option est spécifiée, DynamoDB Local n'envoie aucune donnée de télémétrie.
- `-version` : imprime la version de DynamoDB local.

Définition du point de terminaison local

Par défaut, les outils AWS SDKs et utilisent des points de terminaison pour le service Web Amazon DynamoDB. Pour utiliser les outils SDKs et avec la version téléchargeable de DynamoDB, vous devez spécifier le point de terminaison local :

`http://localhost:8000`

AWS Command Line Interface

Vous pouvez utiliser le AWS Command Line Interface(AWS CLI) pour interagir avec DynamoDB téléchargeable.

Pour accéder à DynamoDB s'exécutant localement, utilisez le paramètre `--endpoint-url`. Voici un exemple d'utilisation du AWS CLI pour répertorier les tables de DynamoDB sur votre ordinateur.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> **Note**
>
> AWS CLI Impossible d'utiliser la version téléchargeable de DynamoDB comme point de terminaison par défaut. Par conséquent, vous devez le spécifier `--endpoint-url` à chaque AWS CLI commande.

AWS SDKs

La façon dont vous spécifiez un point de terminaison dépend du langage de programmation et du kit SDK AWS que vous utilisez. Les sections suivantes décrivent comment le faire :

- Java : définition de la région AWS et du point de terminaison(DynamoDB local prend en charge le SDK AWS pour Java V1 et V2)
- CodeSamples.Java. RegionAndEndpoint .NET : définition de la région AWS et du point de terminaison

Différences entre la version téléchargeable de DynamoDB et le service web DynamoDB

La version téléchargeable de DynamoDB est destinée uniquement à des fins de développement et de test. En comparaison, le service web DynamoDB est un service géré doté de fonctions de mise à l'échelle, de disponibilité et de durabilité, qui en font une solution idéale pour un usage en mode production.

Les différences entre la version téléchargeable de DynamoDB et le service web sont les suivantes :

- Régions AWS et distincts ne Comptes AWS sont pas pris en charge au niveau du client.
- Les paramètres de débit approvisionné sont ignorés dans la version téléchargeable de DynamoDB, même si l'opération `CreateTable` les requiert. Pour `CreateTable`, vous pouvez spécifier les nombres de votre choix pour le débit provisionné en lecture et en écriture, même si ces chiffres ne sont pas utilisés. Vous pouvez appeler `UpdateTable` autant de fois que vous le souhaitez par jour. Cependant, les modifications des valeurs du débit provisionné sont ignorées.
- Les opérations `Scan` sont exécutées de façon séquentielle. Les analyses parallèles ne sont pas prises en charge. Les paramètres `Segment` et `TotalSegments` de l'opération `Scan` sont ignorés.
- La vitesse des opérations de lecture et d'écriture sur les données de la table est limitée uniquement par la vitesse de votre ordinateur. Les opérations `CreateTable`, `UpdateTable` et `DeleteTable` se produisent immédiatement, et l'état de la table est toujours ACTIVE. Les

opérations `UpdateTable` qui modifient uniquement les paramètres de débit provisionné sur les tables ou les index secondaires globaux se produisent immédiatement. Si une opération `UpdateTable` crée ou supprime des index secondaires globaux, ces index passent par les états normaux (tels que CREATING et DELETING, respectivement) avant d'acquérir l'état ACTIVE. La table demeure ACTIVE pendant ce temps.

- Les opérations de lecture sont éventuellement cohérentes. Cependant, en raison de la vitesse d'exécution de DynamoDB local sur votre ordinateur, la plupart des lectures apparaissent fortement cohérentes.
- Les métriques des collections d'élément ne sont pas suivies, ni leurs tailles. Dans les réponses des opérations, les valeurs null sont retournées à la place des métriques des collections d'élément.
- Dans DynamoDB, les données retournées sont limitées à 1 Mo par jeu de résultats. Tant le service web que la version téléchargeable de DynamoDB appliquent cette limite. Cependant, lorsque vous interrogez un index, le service DynamoDB calcule uniquement la taille de la clé et des attributs projetés. En revanche, la version téléchargeable de DynamoDB calcule la taille de l'élément entier.
- Si vous utilisez DynamoDB Streams, la vitesse de création des partitions peut varier. Dans le service web DynamoDB, le comportement de création de partition est partiellement influencé par l'activité de partition de table. Lorsque vous exécutez DynamoDB localement, il n'existe aucun partitionnement de table. Dans les deux cas, comme les partitions sont éphémères, votre application ne doit pas être dépendante du comportement des partitions.
- `TransactionConflictExceptions`ne sont pas lancés par DynamoDB téléchargeable pour les transactions. APIs Nous vous recommandons d'utiliser une infrastructure de simulation Java pour simuler des `TransactionConflictExceptions` dans le gestionnaire DynamoDB pour tester comment votre application répond dans le cas de transactions en conflit.
- Dans le service Web DynamoDB, qu'il soit accessible via la console ou le, les noms des tables distinguent AWS CLI les majuscules et minuscules. Une table nommée `Authors` et une table nommée `authors` peuvent toutes deux exister comme tables distinctes. Dans la version téléchargeable, les noms de table ne sont pas sensibles à la casse, et la tentative de créer ces deux tables se traduira par une erreur.
- L'ajout de balises n'est pas pris en charge dans la version téléchargeable de DynamoDB.
- La version téléchargeable de DynamoDB ignore le paramètre Limit dans. ExecuteStatement
- Les clés à attributs multiples pour l'index secondaire global ne sont pas prises en charge.

Historique des versions de DynamoDB local

Le tableau suivant décrit les modifications importantes apportées à chaque version de DynamoDB local.

Version	Modifier	Description	Date
3.1.0	Amélioration des performances pour les requêtes PartiQL, y compris la dépendance au temps de Joda	• Mise à jour de la version Java du kit SDK vers la dernière version publique de 2.25.50 à 2.33.0 • Inclusion de la dépendance au temps de Joda dans le fichier Pom.xml • Amélioration des performances pour les requêtes PartiQL • Mise à niveau des dépendances pour corriger plusieurs problèmes de vulnérabilité CVE	14 septembre 2025
3.0.0	Migration du AWS SDK Java V1 vers le V2	• Migration du AWS SDK Java V1 vers le V2 • Structure de package mise à jour de `com.amazonaws.services.dyna`	17 juillet 2025

Version	Modifier	Description	Date
		`modbv2` vers `software.amazon.dynamodb.services` • AWS Dépendances supprimées du SDK Java V1	

Version	Modifier	Description	Date
2.6.0	Support de l'ARN de table en tant que nom de table dans DynamoDB APIs Correction des performances et mises à jour de sécurité	• Ajout de la prise en charge de l'utilisation de l'ARN de table comme nom de table dans plusieurs DynamoDB APIs • Correction d'un bug `CreateStreamTable` sur les machines hautement performantes, comme le Mac M3 • Mise à niveau des dépendances pour corriger les problèmes de vulnérabilité (CVE-2022-49043,CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	13 mars 2025

Version	Modifier	Description	Date
2.5.4	Mise à niveau vers Jetty Dependencies	• Mise à niveau de Jetty 12.0.8 vers Jetty 12.0.14 (résout CVE-2024-6763, CVE-2024-8184, CVE-2024-47535) Atténuation pour (CVE-2024-21634)	12 décembre 2024
2.5.3	Mise à niveau de Jackson Dependencies vers la version 2.17.x dans Log4j Core (résout CVE-2022-1471)	• Mise à niveau de Jackson Dependencies vers la version 2.17.x dans Log4j Core (résout CVE-2022-1471) pour corriger une vulnérabilité critique dans la bibliothèque SnakeYAML, qui est une dépendance transitive	6 novembre 2024
2.5.2	Correctif de bogue pour le flux de travail de mise à jour des tables	• Correctif d'un bogue pour le flux de travail lorsque la mise à jour des tables essaie de mettre à jour la table avec le mode facturation à la demande en mode provisionné avec GSI	20 juin 2024

Version	Modifier	Description	Date
2.5.1	Correctif pour les bogues introduits dans la fonctionnalité `OndemandThroughPut`	• Correctif de quelques bogues liés à `OndemandThroughPut`	5 juin 2024
2.5.0	Prise en charge d'un débit maximal configurable pour les tables à la demande, `ReturnValuesOnConditionCheckFailure` `BatchExecuteStatement` et `ExecuteTransaction` `Request`	• Ajout de la télémétrie au mode intégré • Corriger la SDKv2 traduction pour `ConditionalCheckException`	28 mai 2024
2.4.0	Prise en charge pour `ReturnValuesOnConditionCheckFailure` - Mode intégré	• Correctif de mode intégré pour `TrimmedDataAccessException` pour le fonctionnement sur plusieurs flux • Corriger la traduction des exceptions SDKv2 en mode intégré	17 avril 2024

Version	Modifier	Description	Date
2.3.0	Mise à niveau de Jetty et de JDK	• Mise à niveau vers Jetty 12.0.2 • Mise à niveau vers le kit JDK 17 • Mise à niveau ANTLR4 vers la version 4.10.1	14 mars 2024
2.2.0	Ajout de la prise en charge de la protection contre la suppression des tables et du paramètre `ReturnValuesOnConditionCheckFailure`	• Ajout de la prise en charge de la protection contre la suppression des tables • Support supplémentaire pour ReturnValuesOnConditionCheckFailure • Ajouté de la prise en charge de l’indicateur -version	14 décembre 2023

Version	Modifier	Description	Date
2.1.0	Support des bibliothèques SQLLite natives pour les projets Maven et ajout de la télémétrie	• Ajout de la télémétrie à DynamoDB Local • Copiez dynamiquement les bibliothèques SQLLite natives pour les projets Maven • Retrait de la bibliothèque io.github.ganadist.sqlite4java de la dépendance Maven • Mise à niveau GoogleGuava vers 32.1.1-jre	23 octobre 2023

Version	Modifier	Description	Date
2.0.0	Migration de Javax vers Jakarta Namespace et Support JDK11	• Migration de l'espace de noms et du support de Javax vers Jakarta JDK11 • Correction de la gestion de l’accès et de la clé secrète non valides lors du démarrage du serveur • Correction des vulnérabilités identifiées dans Maven en mettant à jour les dépendanc es	5 juillet 2023
1.25.1	Mise à niveau de Jackson Dependenc ies vers la version 2.17.x dans Log4j Core (résout CVE-2022-1471)	Mise à niveau de Jackson Dependenc ies vers la version 2.17.x dans Log4j Core (résout CVE-2022-1471) pour corriger une vulnérabilité critique dans la bibliothèque SnakeYAML, qui est une dépendance transitive	6 novembre 2024

Version	Modifier	Description	Date
1.25.0	Ajout de la prise en charge de la protection contre la suppression des tables et du paramètre `ReturnValuesOnConditionCheckFailure`	• Ajout de la prise en charge de la protection contre la suppression des tables • Support supplémentaire pour ReturnValuesOnConditionCheckFailure • Ajouté de la prise en charge de l'indicateur -version	18 décembre 2023
1.24.0	Support des bibliothèques SQLLite natives pour les projets Maven et ajout de la télémétrie	• Ajout de la télémétrie à DynamoDB Local • Copiez dynamiquement les bibliothèques SQLLite natives pour les projets Maven • Retrait de la bibliothèque io.github.ganadist.sqlite4java de la dépendance Maven • Mise à niveau GoogleGuava vers 32.1.1-jre	23 octobre 2023

Version	Modifier	Description	Date
1.23.0	Gestion de l'accès et de la clé secrète non valides lors du démarrage du serveur	• Correction de la gestion de l'accès et de la clé secrète non valides lors du démarrage du serveur • Correction des vulnérabilités identifiées dans Maven en mettant à jour les dépendances	28 juin 2023
1.22.0	Prise en charge de l'opération Limit pour PartiQL	• Optimisation de la clause IN pour PartiQL • Prise en charge de l'opération Limit • Prise en charge de M1 pour les projets Maven	8 juin 2023

Version	Modifier	Description	Date
1.21.0	Prise en charge de 100 actions par transaction	• Augmentation du nombre d'actions par transaction de 25 à 100 • Mise à niveau de l'image Docker Open JDK vers la version 11 • Fixation de la parité pour les exceptions déclenchées en cas de duplication d'éléments dans BatchExecuteStatement	26 janvier 2023
1.20.0	Ajout de la prise en charge de M1 Mac	• Ajout de la prise en charge de M1 Mac • Mise à niveau de la dépendance Jetty vers la version 9.4.48.v20220622	12 septembre 2022
1.19.0	Mise à niveau de l'analyseur PartiQL	Mise à niveau de l'analyseur PartiQL et d'autres bibliothèques associées	27 juillet 2022
1.18.0	Mise à niveau de log4j-core et Jackson-core	Mise à niveau de log4j-core vers la version 2.17.1 et de Jackson-core 2.10.x vers la version 2.12.0	10 janvier 2022

Version	Modifier	Description	Date
1.17.2	Mise à niveau de log4j-core	Mise à niveau de la dépendance log4j-core vers la version 2.16	16 janvier 2021
1.17.1	Mise à niveau de log4j-core	Mise à jour de la dépendance log4j-core pour corriger l'exploit « jour zéro » afin d'empêcher l'exécution du code à distance - Log4Shel	10 janvier 2021
1.17.0	Obsolescence du shell Web Javascript	• Mise à jour de la dépendance du AWS SDK vers AWS SDK for Java 1.12.x • Obsolescence du shell Web Javascript	8 janvier 2021

Télémétrie dans DynamoDB Local

Chez AWS, nous développons et lançons des services basés sur ce que nous apprenons des interactions avec les clients et nous utilisons les commentaires des clients pour améliorer nos produits. La télémétrie est une information supplémentaire qui nous aide à mieux comprendre les besoins de nos clients, à diagnostiquer les problèmes et à fournir des fonctionnalités permettant d'améliorer l'expérience client.

DynamoDB Local collecte des données de télémétrie, telles que les métriques d'utilisation génériques, les informations relatives aux systèmes et à l'environnement et les erreurs. Pour plus de détails sur les types de télémétrie collectés, consultez Type d'informations à collecter.

DynamoDB Local ne collecte pas les informations personnelles, telles que les noms d'utilisateur ou les adresses e-mail. Elle n'extrait pas non plus les informations sensibles au niveau du projet.

En tant que client, vous contrôlez si la télémétrie est activée et vous pouvez modifier vos paramètres à tout moment. Si la télémétrie reste activée, DynamoDB Local envoie des données de télémétrie en arrière-plan sans nécessiter d'interaction supplémentaire avec le client.

Désactivation de la télémétrie à l'aide des options de ligne de commande

Vous pouvez désactiver la télémétrie à l'aide des options de ligne de commande au démarrage de DynamoDB Local à l'aide de l'option `-disableTelemetry`. Pour plus d'informations, consultez Options de ligne de commande .

Désactivation de la télémétrie pour une session unique

Dans les systèmes d'exploitation macOS et Linux, vous pouvez désactiver la télémétrie pour une seule session. Pour désactiver la télémétrie de votre session en cours, exécutez la commande suivante pour définir la variable d'environnement `DDB_LOCAL_TELEMETRY` sur `false`. Répétez la commande pour chaque nouveau terminal ou chaque nouvelle session.

```
export DDB_LOCAL_TELEMETRY=0
```

Désactivation de la télémétrie pour votre profil dans toutes les sessions

Exécutez les commandes suivantes pour désactiver la télémétrie pour toutes les sessions lorsque vous exécutez DynamoDB Local sur votre système d'exploitation.

Pour désactiver la télémétrie sous Linux

1. Exécuter :

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. Exécuter :

   ```
   source ~/.profile
   ```

Pour désactiver la télémétrie sous macOS

1. Exécuter :

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. Exécuter :

```
source ~/.profile
```

Pour désactiver la télémétrie sous Windows

1. Exécuter :

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Exécuter :

```
refreshenv
```

Désactivation de la télémétrie à l'aide de DynamoDB local intégré aux projets Maven

Vous pouvez désactiver la télémétrie à l'aide de DynamoDB local intégré aux projets Maven.

```
boolean disableTelemetry = true;
// AWS SDK v1
 AmazonDynamoDB amazonDynamoDB =
 DynamoDBEmbedded.create(disableTelemetry).amazonDynamoDB();

// AWS SDK v2
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

Type d'informations à collecter

- Informations d'utilisation : données de télémétrie génériques, telles que le démarrage/arrêt du serveur et l'API ou l'opération appelée.
- Informations sur le système et l'environnement : la version de Java, le système d'exploitation (Windows, Linux ou macOS), l'environnement dans lequel DynamoDB Local s'exécute (par exemple, fichier JAR autonome, conteneur Docker ou dépendance Maven) et les valeurs de hachage des attributs d'utilisation.

En savoir plus

Les données de télémétrie collectées par DynamoDB Local sont conformes aux politiques de confidentialité des données AWS. Pour plus d'informations, consultez les ressources suivantes :

- Conditions générales du service AWS
- Questions fréquentes (FAQ) relatives à la confidentialité des données

Étape 1 : création d'une table dans DynamoDB

Dans cette étape, vous allez créer une table `Music` dans Amazon DynamoDB. Cette table possède les informations suivantes :

- Clé de partition : `Artist`
- Clé de tri : `SongTitle`

Pour plus d'informations sur les opérations de table, consultez Utilisation de tables et de données dans DynamoDB.

> **Note**
>
> Avant de commencer, assurez-vous d'avoir suivi les étapes dans Conditions préalables.

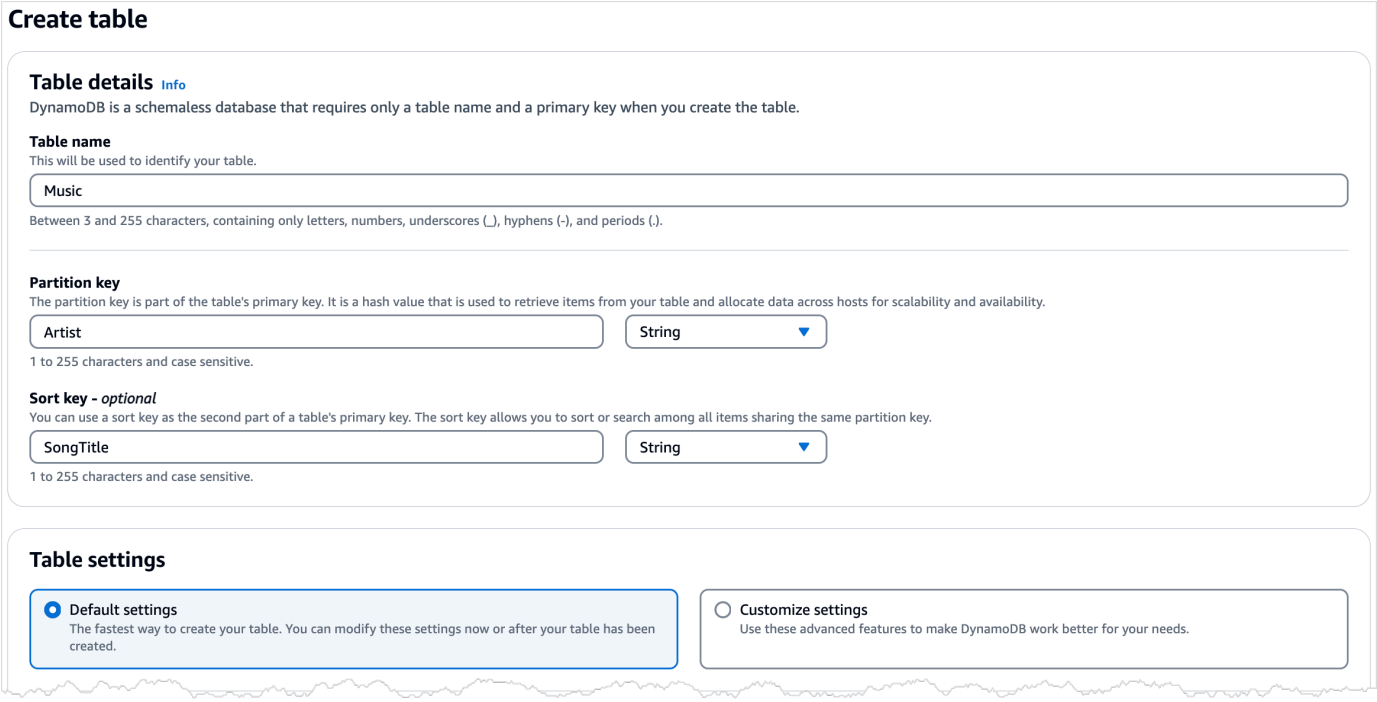
AWS Management Console

Pour créer une table `Music` à l'aide de la console DynamoDB :

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Choisissez Créer un tableau.
4. Saisissez les informations de la table comme suit :
   a. Sous Nom du tableau, saisissez **`Music`**.
   b. Pour Clé de partition, saisissez **`Artist`**.

c. Pour Clé de tri, entrez **`SongTitle`**.

5. Pour Paramètres de la table, conservez la sélection par défaut de Paramètres par défaut.
6. Choisissez Créer une table pour créer la table.



7. Une fois que la table a le statut `ACTIVE`, nous vous recommandons d'activer Sauvegardes ponctuelles pour DynamoDB sur cette table en exécutant les étapes suivantes :

   a. Cliquez sur le nom de la table pour l'ouvrir.

   b. Choisissez Sauvegarde.

   c. Choisissez Modifier dans la section Point-in-time de restauration (PITR).

   d. Sur la page Modifier les paramètres de point-in-time restauration, choisissez Activer la point-in-time restauration.

   e. Sélectionnez Enregistrer les modifications.

AWS CLI

L'AWS CLI exemple suivant crée une nouvelle `Music` table à l'aide de`create-table`.

Linux

```
aws dynamodb create-table \
    --table-name Music \
```

```
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows CMD

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

L'utilisation de `create-table` renvoie l'exemple de résultat suivant.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

Veuillez noter que la valeur du champ `TableStatus` est définie sur `CREATING`.

Pour vérifier que DynamoDB a fini de créer la table `Music`, utilisez la commande `describe-table`.

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

Windows CMD

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

Cette commande renvoie le résultat suivant. Lorsque DynamoDB finit de créer la table, la valeur du champ `TableStatus` est définie sur `ACTIVE`.

```
"TableStatus": "ACTIVE",
```

Une fois que la table a le statut `ACTIVE`, il est considéré comme une bonne pratique d'activer Sauvegardes ponctuelles pour DynamoDB sur cette table en exécutant les commandes suivantes.

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

Windows CMD

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-
specification PointInTimeRecoveryEnabled=true
```

Cette commande renvoie le résultat suivant.

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> **Note**
>
> L'activation de sauvegardes continues avec point-in-time restauration entraîne des implications financières. Pour plus d'informations sur la tarification, veuillez consulter Tarification Amazon DynamoDB.

AWS SDK

Les exemples de code suivants montrent comment créer une table DynamoDB à l'aide d'un kit AWS SDK.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
#####################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#####################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#####################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#####################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#####################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#####################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, voir CreateTablela section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour créer une table avec des balises

L'exemple `create-table` suivant utilise les attributs et le schéma de clés spécifiés pour créer une table nommée `MusicCollection`. Cette table utilise le débit provisionné et est chiffrée au repos à l'aide de la clé CMK AWS détenue par défaut. La commande applique également une balise à la table, avec une clé `Owner` et une valeur `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
```

```
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour créer une table en mode à la demande

L'exemple suivant crée une table appelée `MusicCollection` en mode à la demande, plutôt qu'en mode débit provisionné. Cela est utile pour les tables dont les charges de travail sont imprévisibles.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```


Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour créer une table et la chiffrer à l'aide d'une clé CMK gérée par le client

L'exemple suivant crée une table nommée `MusicCollection` et la chiffre à l'aide d'une clé CMK gérée par le client.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
```

```
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour créer une table avec un index secondaire local

L'exemple suivant utilise les attributs et le schéma de clés spécifiés pour créer une table nommée `MusicCollection` avec un index local secondaire nommé `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
```

```
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 5 : pour créer une table avec un index secondaire global

L'exemple suivant crée une table nommée `GameScores` avec un index secondaire global nommé `GameTitleIndex`. La table de base a une clé de partition `UserId` et une clé de tri `GameTitle`, vous permettant de trouver efficacement le meilleur score d'un utilisateur pour un jeu spécifique, tandis que l'index secondaire global (GSI) a une clé de partition `GameTitle` et

une clé de tri `TopScore`, vous permettant de trouver rapidement le score le plus élevé pour un jeu particulier.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
```

```
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
```

```
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 6 : pour créer une table avec plusieurs index secondaires globaux à la fois

L'exemple suivant crée une table nommée `GameScores` avec deux index secondaires globaux. Les schémas GSI sont transmis via un fichier plutôt que sur la ligne de commande.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Contenu de `gsi.json` :

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
```

```
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
```

```
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
```

```
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 7 : pour créer une table avec Streams activé

L'exemple suivant crée une table appelée `GameScores` avec DynamoDB Streams activé. Les nouvelles et les anciennes images de chaque élément seront écrites dans le flux.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
```

```
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 8 : pour créer une table avec Keys-Only Stream activé

L'exemple suivant crée une table appelée `GameScores` avec DynamoDB Streams activé. Seuls les attributs clés des éléments modifiés sont écrits dans le flux.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
```

```
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
```

```
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Pour plus d'informations, consultez Modifier la récupération de données pour DynamoDB Streams dans le Guide du développeur Amazon DynamoDB.

Exemple 9 : pour créer une table à l'aide de la classe Standard Infrequent Access

L'exemple suivant crée une table appelée `GameScores` et attribue la classe de table Standard-Infrequent Access (DynamoDB Standard-IA). Cette classe de table est optimisée pour le stockage, qui constitue le principal coût.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
```

```
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Pour plus d'informations, consultez Classes de tables dans le Guide du développeur Amazon DynamoDB.

Exemple 10 : pour créer une table avec la protection contre la suppression activée

L'exemple suivant crée une table appelée `GameScores` et active la protection contre la suppression.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
```

```
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Pour plus d'informations, consultez Utilisation de la protection contre la suppression dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, voir CreateTablela section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
```

```
   public static void main(String[] args) {
       final String usage = """

           Usage:
               <tableName> <key>

           Where:
               tableName - The Amazon DynamoDB table to create (for example,
Music3).
               key - The key for the Amazon DynamoDB table (for example,
Artist).
           """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
           .region(region)
           .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
           .attributeDefinitions(AttributeDefinition.builder()
               .attributeName(key)
               .attributeType(ScalarAttributeType.S)
               .build())
           .keySchema(KeySchemaElement.builder()
               .attributeName(key)
               .keyType(KeyType.HASH)
               .build())
```

```
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
```

```
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
```

```
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Pour plus de détails sur l'API, consultez CreateTablela section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créer une table.

```
        $tableName = "ddb_demo_table_$uuid";
```

```
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Le

schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemple 2 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Un index secondaire local est également défini. La clé de l'index secondaire local sera définie automatiquement à partir de la clé de hachage principale de la table (ForumName). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
```

```
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemple 3 : Cet exemple montre comment utiliser un pipeline unique pour créer une table nommée Thread qui possède une clé primaire composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés) et un index secondaire local. Les options Add- DDBKey Schema et Add- DDBIndex Schema créent un nouvel TableSchema objet pour vous si aucun objet n'est fourni par le pipeline ou le paramètre -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemple 2 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Un index secondaire local est également défini. La clé de l'index secondaire local sera définie automatiquement à partir de la clé de hachage principale de la table (ForumName). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemple 3 : Cet exemple montre comment utiliser un pipeline unique pour créer une table nommée Thread qui possède une clé primaire composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés) et un index secondaire local. Les options Add- DDBKey Schema et Add- DDBIndex Schema créent un nouvel TableSchema objet pour vous si aucun objet n'est fourni par le pipeline ou le paramètre -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

Créez une table pour stocker des données vidéo.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
```

```
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Pour plus de détails sur l'API, consultez CreateTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
```

```
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Pour plus de détails sur l'API, voir CreateTablela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
```

```
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez CreateTablela section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
```

```
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir plus d’exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Une fois la nouvelle table créée, passez à l’ Étape 2 : écrire des données dans une table DynamoDB.

Étape 2 : écrire des données dans une table DynamoDB

Dans cette étape, vous insérez plusieurs éléments dans la table `Music` que vous avez créée dans Étape 1 : création d'une table dans DynamoDB.

Pour plus d'informations sur les opérations d'écriture, consultez Écriture d'un élément.

AWS Management Console

Suivez ces étapes pour écrire des données sur la table `Music` à l'aide de la console DynamoDB.

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Sur la page Tables, sélectionnez la table Musique.
4. Sélectionnez Explorer les éléments de table.
5. Dans la section Éléments retournés, choisissez Créer un article.
6. Sur la page Créer un article, procédez comme suit pour ajouter des éléments à la table :
    a. Choisissez Ajouter un nouvel attribut, puis choisissez Numéro.
    b. Dans Nom de l'attribut, entrez **`Awards`**.
    c. Répétez cette procédure pour créer un **`AlbumTitle`** de type Chaîne.
    d. Saisissez les valeurs suivantes pour votre élément :
        i. Pour Artist (Artiste), saisissez **`No One You Know`**.
        ii. Pour SongTitle, saisissez **`Call Me Today`**.
        iii. Pour AlbumTitle, saisissez **`Somewhat Famous`**.
        iv. Pour Awards (Récompenses), saisissez **`1`**.
7. Choisissez Créer un élément.
8. Répétez cette procédure et créez un autre élément avec les valeurs suivantes :
    a. Pour Artist (Artiste), saisissez **`Acme Band`**.
    b. Pour SongTitleentrer**`Happy Day`**.
    c. Pour AlbumTitle, saisissez **`Songs About Life`**.
    d. Pour Awards (Récompenses), saisissez **`10`**.
9. Effectuez cette opération de nouveau pour créer un autre élément avec le même Artist (Artiste) qu'à l'étape précédente, mais avec des valeurs différentes pour les autres attributs :

a. Pour Artist (Artiste), saisissez **`Acme Band`**.

b. Pour SongTitleentrer**`PartiQL Rocks`**.

c. Pour AlbumTitle, saisissez **`Another Album Title`**.

d. Pour Awards (Récompenses), saisissez **`8`**.

AWS CLI

L'AWS CLI exemple suivant crée plusieurs nouveaux éléments dans le `Music` tableau. Vous pouvez procéder via l'API DynamoDB ou PartiQL, un langage de requête compatible SQL pour DynamoDB.

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

Windows CMD

```
aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"
```

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
               {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

Windows CMD

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

Pour plus d'informations sur l'écriture de données avec PartiQL, veuillez consulter Instructions d'insertion de PartiQL.

Pour plus d'informations sur les types de données pris en charge dans DynamoDB, consultez Types de données.

Pour plus d'informations sur la façon de représenter les types de données DynamoDB dans JSON, consultez Valeurs d'attributs.

AWS SDK

Les exemples de code suivants montrent comment écrire un élément dans une table DynamoDB à l'aide d'un SDK.AWS

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
```

```
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
```

```
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
#########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, voir PutItemla section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);
```

```
    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
```

```
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour ajouter un élément à une table

L'`put-item`exemple suivant ajoute un nouvel élément au MusicCollectiontableau.

```
aws dynamodb put-item \
```

```
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenu de `item.json` :

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Sortie :

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour remplacer un élément d'une table sous certaines conditions

L'exemple `put-item` suivant remplace un élément existant dans la table `MusicCollection` uniquement si celui-ci possède un attribut `AlbumTitle` dont la valeur est `Greatest Hits`. La commande renvoie la valeur précédente de l'élément.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenu de `item.json` :

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Contenu de `names.json` :

```
{
    "#A": "AlbumTitle"
}
```

Contenu de `values.json` :

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Sortie :

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
```

```
    }
}
```

Si la clé existe déjà, vous devriez voir la sortie suivante :

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, voir PutItemla section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
```

```
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Met un élément dans un tableau en utilisant DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
```

```
                    tableName - The Amazon DynamoDB table in which an item is
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }
```

```
    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Insérez un élément dans la table.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Insérez un élément dans une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
```

```
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Pour plus de détails sur l'API, consultez PutItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        echo "What's the name of the last movie you watched?\n";
```

```
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : création d'un nouvel élément ou remplacement d'un élément existant par un nouvel élément.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
```

```
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : création d'un nouvel élément ou remplacement d'un élément existant par un nouvel élément.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
```

```
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour plus de détails sur l'API, consultez PutItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
```

```
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);
```

```
    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Pour plus de détails sur l'API, voir PutItemla section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez PutItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }
```

```
            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
```

```
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir plus d'exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Après avoir écrit des données dans votre table, passez à l' Étape 3 : lire des données à partir d'une table DynamoDB.

Étape 3 : lire des données à partir d'une table DynamoDB

Dans cette étape, vous relirez l'un des éléments que vous avez créé dans Étape 2 : écrire des données dans une table DynamoDB. Vous pouvez utiliser la console DynamoDB ou AWS CLI le pour lire un élément du `Music` tableau en spécifiant et. `Artist` `SongTitle`

Pour plus d'informations sur les opérations de lecture dans DynamoDB, consultez Lecture d'un élément.

AWS Management Console

Suivez ces étapes pour lire des données de la table `Music` à l'aide de la console DynamoDB.

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Sur la page Tables, sélectionnez la table Musique.
4. Sélectionnez Explorer les éléments de table.
5. Dans la section Éléments retournés, affichez la liste d'éléments stockés dans la table, triés par `Artist` et `SongTitle`. Le premier élément de la liste est celui avec l'artiste Acme Band et les SongTitlePartiQL Rocks.

AWS CLI

L'AWS CLI exemple suivant lit un élément à partir du`Music`. Vous pouvez procéder via l'API DynamoDB ou PartiQL, un langage de requête compatible SQL pour DynamoDB.

DynamoDB API

> **Note**
>
> Le comportement par défaut pour DynamoDB est la lecture éventuellement cohérente. Le paramètre `consistent-read` est utilisé ci-dessous pour illustrer des lectures cohérentes fortes.

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

Windows CMD

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

L'utilisation de `get-item` renvoie l'exemple de résultat suivant.

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
```

```
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

L'utilisation de l'instruction PartiQL `Select` renvoie l'exemple de résultat suivant.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Pour plus d'informations sur la lecture de données avec PartiQL, consultez Instructions de sélection de PartiQL.

AWS SDK

Les exemples de code suivants montrent comment lire un élément d'une table DynamoDB à l'aide d'un SDK.AWS

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };
```

```
            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
```

```
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
```

```
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
query inserts on some strings.
  else
    echo "$response"
```

```
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
```

```
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, voir GetItemla section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;
```

```
    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour lire un élément dans une table

L'exemple `get-item` suivant récupère un élément de la table `MusicCollection`. La table possède une clé hash-and-range primaire (`ArtistetSongTitle`), vous devez donc spécifier

ces deux attributs. La commande demande également des informations sur la capacité de lecture consommée par l'opération.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Sortie :

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour lire un élément en utilisant une lecture cohérente

L'exemple suivant récupère un élément à partir de la table `MusicCollection` à l'aide de la lecture fortement cohérente.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Sortie :

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour extraire des attributs spécifiques d'un élément

L'exemple suivant utilise une expression de projection pour extraire uniquement trois attributs de l'élément souhaité.

```
aws dynamodb get-item \
```

```
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Contenu de `names.json` :

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Sortie :

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, voir GetItemla section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
```

```
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
```

```
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Récupère un élément d'une table à l'aide du DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
 tableName);
        Region region = Region.US_EAST_1;
```

```
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir un élément d'une table

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenez un élément d'une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
```

```
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Pour plus de détails sur l'API, consultez GetItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : renvoie l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : renvoie l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem
```

```
Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
```

```
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Pour plus de détails sur l'API, consultez GetItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        oo_item = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = it_key ).
        DATA(lt_attr) = oo_item->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez GetItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB
```

```
    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir plus d'exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Pour mettre à jour les données dans votre table, passez à l' Étape 4 : mettre à jour des données dans une table DynamoDB.

Étape 4 : mettre à jour des données dans une table DynamoDB

Dans cette étape, vous mettez à jour un élément que vous avez créé dans Étape 2 : écrire des données dans une table DynamoDB. Vous pouvez utiliser la console DynamoDB ou AWS CLI le pour mettre à jour un élément `AlbumTitle` du tableau en `Music` `Artist` spécifiant`SongTitle`, et en mettant à jour. `AlbumTitle`

Pour plus d'informations sur les opérations d'écriture, consultez Écriture d'un élément.

AWS Management Console

Vous pouvez utiliser la console DynamoDB pour mettre à jour des données dans la table `Music`.

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Dans la liste de tables, choisissez la table Music (Musique).
4. Sélectionnez Explorer les éléments de table.
5. Dans Articles retournés, pour la ligne d'articles avec Acme Band Artist et Happy Day SongTitle, procédez comme suit :

    a. Placez votre curseur sur le AlbumTitletitre Songs About Life.
    b. Choisissez l'icône Modifier.
    c. Dans la fenêtre contextuelle Modifier la chaîne, entrez **`Songs of Twilight`**.
    d. Choisissez Enregistrer.

> **Tip**
>
> Pour pouvez aussi, pour mettre à jour un élément, procéder comme suit dans la section Articles retournés :
>
> 1. Choisissez la rangée d'articles avec l'artiste Acme Band et SongTitleHappy Day.

2. Dans la liste déroulante Actions, choisissez Modifier l'élément.
3. Pour entrer AlbumTitle, entrez**`Songs of Twilight`**.
4. Choisissez Enregistrer et fermer.

AWS CLI

L'AWS CLI exemple suivant met à jour un élément du `Music` tableau. Vous pouvez procéder via l'API DynamoDB ou PartiQL, un langage de requête compatible SQL pour DynamoDB.

DynamoDB API

Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

Windows CMD

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
    --return-values ALL_NEW
```

L'utilisation de `update-item` renvoie l'exemple de résultat suivant car `return-values` `ALL_NEW` a été spécifié.

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
```

```
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

Windows CMD

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

L'utilisation de l'instruction `Update` renvoie l'exemple de résultat suivant car `RETURNING ALL NEW *` a été spécifié.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
```

```
            }
        }
    ]
}
```

Pour plus d'informations sur la mise à jour des données avec PartiQL, consultez Instructions de mise à jour de PartiQL.

AWS SDK

Les exemples de code suivants montrent comment mettre à jour un élément dans une table DynamoDB à l'aide d'un SDK.AWS

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
```

```
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
```

```
 local table_name keys update_expression values response
 local option OPTARG # Required to use getopts command in a function.

 ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_update_item"
   echo "Update an item in a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k keys  -- Path to json file containing the keys that identify the
item to update."
   echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
   echo " -v values  -- Path to json file containing the update values."
   echo ""
 }

 while getopts "n:k:e:v:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     e) update_expression="${OPTARG}" ;;
     v) values="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
```

```
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Fonctions utilitaires utilisées dans cet exemple.

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, voir UpdateItemla section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */
```

```
/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
```

```
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
```

```
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour mettre à jour un élément dans une table

L'exemple `update-item` suivant met à jour un élément dans la table `MusicCollection`. Il ajoute un nouvel attribut (`Year`) et modifie l'attribut `AlbumTitle`. Tous les attributs de l'élément, tels qu'ils apparaissent après la mise à jour, sont renvoyés dans la réponse.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenu de `key.json` :

```
{
```

```
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenu de `expression-attribute-names.json` :

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contenu de `expression-attribute-values.json` :

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Sortie :

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
```

```
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour mettre à jour un élément sous certaines conditions

L'exemple suivant met à jour un élément de la table `MusicCollection`, mais uniquement si l'élément existant ne possède pas encore d'attribut `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenu de `expression-attribute-names.json` :

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenu de `expression-attribute-values.json` :

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Si l'élément possède déjà un attribut `Year`, DynamoDB renvoie le résultat suivant.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, voir UpdateItemla section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
```

```
  }
 }
 return attributeMap, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
```

```
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Met à jour un élément d'un tableau à l'aide de DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
```

```
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Pour plus de détails sur l'API, consultez UpdateItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
```

```
            ],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : définit l'attribut genre sur « Rap » sur l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Rap
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : définit l'attribut genre sur « Rap » sur l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Rap
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Mettez à jour un élément à l'aide d'une expression de mise à jour.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
```

```
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Mettez à jour un élément à l'aide d'une expression de mise à jour qui inclut une opération arithmétique.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
```

```
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Mettre à jour un élément uniquement lorsqu'il remplit certaines conditions.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
```

```
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Attributes"]
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
 #{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez UpdateItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
```

```
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
```

```
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir plus d'exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Pour interroger les données dans la table `Music`, passez à l'Étape 5 : interroger des données dans une table DynamoDB.

Étape 5 : interroger des données dans une table DynamoDB

Dans cette étape, vous interrogez les données que vous avez écrites dans la table `Music` dans the section called "Étape 2 : écrire des données" en spécifiant l'`Artist`. Cela affichera tous les morceaux associés à la clé de partition : `Artist`.

Pour plus d'informations sur les opérations d'interrogation, consultez Interrogation de tables dans DynamoDB.

AWS Management Console

Suivez ces étapes pour utiliser la console DynamoDB afin d'interroger les données de la table `Music`.

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.

3. Dans la liste de tables, choisissez la table Music (Musique).
4. Sélectionnez Explorer les éléments de table.
5. Dans Analyser ou interroger des éléments, assurez-vous que l'option Requête est sélectionnée.
6. Pour Clé de partition, entrez **`Acme Band`**, puis choisissez Exécuter.

AWS CLI

L'AWS CLI exemple suivant interroge un élément de la `Music` table. Vous pouvez procéder via l'API DynamoDB ou PartiQL, un langage de requête compatible SQL pour DynamoDB.

DynamoDB API

Vous interrogez un élément via l'API DynamoDB en utilisant `query` et en fournissant la clé de partition.

Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

Une telle utilisation de `query` renvoie tous les morceaux associés à cet `Artist` spécifique.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
```

```
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

Vous interrogez un élément via PartiQL en utilisant l'instruction `Select` et en fournissant la clé de partition.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

Une telle utilisation de l'instruction `Select` renvoie tous les morceaux associés à cet `Artist` spécifique.


```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```


Pour plus d'informations sur l'interrogation de données avec PartiQL, consultez Instructions de sélection de PartiQL.

AWS SDK

Les exemples de code suivants expliquent comment interroger une table DynamoDB à l'aide d'un kit AWS SDK.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
```

```
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
```

```
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour .NET.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
```

```
 # Function usage explanation
 ####################################
 function usage() {
   echo "function dynamodb_query"
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
```

```
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus d'informations sur l'API, consultez Query dans la Référence des commandes AWS CLI.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */
```

```
bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::QueryRequest request;

   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
```

```
                          << std::endl;
                // Iterate each item and print.
                for (const auto &item: items) {
                    std::cout
                            <<
"*************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour C++.

CLI

AWS CLI

Exemple 1 : pour interroger une table

L'exemple `query` suivant interroge des éléments dans la table `MusicCollection`. La table possède une clé hash-and-range primaire (`ArtistetSongTitle`), mais cette requête indique

uniquement la valeur de la clé de hachage. Elle renvoie les titres des chansons de l'artiste « No One You Know ».

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Pour plus d'informations, consultez Utilisation de requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour interroger une table à l'aide de lectures fortement cohérentes et parcourir l'index par ordre décroissant

L'exemple suivant exécute la même requête que le premier exemple, mais renvoie les résultats dans l'ordre inverse et utilise des lectures fortement cohérentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour filtrer des résultats spécifiques

L'exemple suivant interroge `MusicCollection`, mais exclut les résultats contenant des valeurs spécifiques dans l'attribut `AlbumTitle`. Notez que cela n'affecte pas `ScannedCount` ni `ConsumedCapacity`, car le filtre est appliqué après la lecture des éléments.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Contenu de `values.json` :

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenu de `names.json` :

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Sortie :

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
```

```
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour extraire uniquement le nombre d'éléments

L'exemple suivant extrait le nombre d'éléments correspondant à la requête, mais n'extrait aucun des éléments eux-mêmes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 5 : pour interroger un index

L'exemple suivant interroge l'index secondaire global `AlbumTitleIndex`. La requête renvoie tous les attributs de la table de base qui ont été projetés dans l'index secondaire local. Notez que lorsque vous interrogez un index secondaire local ou un index secondaire global, vous devez également fournir le nom de la table de base à l'aide du paramètre `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
```

```
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

- Pour plus d'informations sur l'API, consultez Query dans la Référence des commandes AWS CLI.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
```

```
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour Go.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Interroge une table à l'aide de DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
```

```
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
```

```
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Interroge une table à l'aide de `DynamoDbClient` et d'un index secondaire.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
```

```
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
```

```
        return response.count
    }
}
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK pour Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
```

```
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour PHP.

PowerShell

Outils pour PowerShell V4

Exemple 1 : invoque une requête qui renvoie des éléments DynamoDB avec les valeurs spécifiées et Artist. SongTitle

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
```

```
AlbumTitle                      Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des applets de commande pour les Outils AWS pour PowerShell (V4).

Outils pour PowerShell V5

Exemple 1 : invoque une requête qui renvoie des éléments DynamoDB avec les valeurs spécifiées et Artist. SongTitle

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Sortie :

```
Name                            Value
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des applets de commande pour les Outils AWS pour PowerShell (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Interrogez des éléments à l'aide d'une expression de condition clé.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

Interrogez des éléments et projetez-les pour renvoyer un sous-ensemble de données.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
```

```
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour Ruby.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Trouvez les films réalisés au cours de l'année spécifiée.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK pour Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
```

```
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Pour plus d'informations sur l'API, consultez Query dans le guide de référence d'API du kit SDK AWS pour SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB
```

```
    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
```

```
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence de l'API du kit SDK AWS for Swift.

Pour obtenir plus d'exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Pour créer un index secondaire global pour votre table, passez à l' Étape 6 : (Facultatif) supprimer votre table DynamoDB pour nettoyer les ressources.

Étape 6 : (Facultatif) supprimer votre table DynamoDB pour nettoyer les ressources

Si vous n'avez plus besoin de la table Amazon DynamoDB que vous avez créée pour le didacticiel, vous pouvez la supprimer. Cette étape permet de vous assurer de ne pas être facturé pour des ressources que vous n'utilisez pas. Vous pouvez utiliser la console DynamoDB ou AWS CLI le pour supprimer la table que vous avez `Music` créée dans. Étape 1 : création d'une table dans DynamoDB

Pour plus d'informations sur les opérations de table dans DynamoDB, consultez Utilisation de tables et de données dans DynamoDB.

AWS Management Console

Pour supprimer la table `Music` à l'aide de la console :

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Choisissez la case à cocher en regard de la table Musique dans la liste de tables.
4. Sélectionnez Delete (Supprimer).

AWS CLI

L'AWS CLI exemple suivant supprime le `Music` tableau à l'aide `delete-table` de.

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

Les exemples de code suivants montrent comment supprimer une table DynamoDB à l'aide d'un kit SDK AWS.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
```

```
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
```

```
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, voir DeleteTablela section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour supprimer une table

L'exemple `delete-table` suivant supprime la table `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Sortie :

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Pour plus d'informations, consultez Suppression d'une table dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, voir DeleteTablela section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
```

```
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
```

```
        println("$tableNameVal was deleted")
    }
}
```

- Pour plus de détails sur l'API, consultez DeleteTablela section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Pour plus de détails sur l'API, voir DeleteTablela section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : supprime la table spécifiée. Vous êtes invité à confirmer avant que l'opération ne se poursuive.

```
Remove-DDBTable -TableName "myTable"
```

Exemple 2 : supprime la table spécifiée. Aucune confirmation ne vous est demandée avant le début de l'opération.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : supprime la table spécifiée. Vous êtes invité à confirmer avant que l'opération ne se poursuive.

```
Remove-DDBTable -TableName "myTable"
```

Exemple 2 : supprime la table spécifiée. Aucune confirmation ne vous est demandée avant le début de l'opération.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
```

```
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour plus de détails sur l'API, consultez DeleteTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, voir DeleteTablela section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Pour plus de détails sur l'API, voir DeleteTablela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
```

```
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez DeleteTablela section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
```

```
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir plus d'exemples avec DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

Poursuivre l'apprentissage concernant DynamoDB

Pour plus d'informations sur l'utilisation d'Amazon DynamoDB, consultez les rubriques suivantes :

- Utilisation de tables et de données dans DynamoDB
- Utilisation d'éléments et d'attributs dans DynamoDB
- Interrogation de tables dans DynamoDB
- Utilisation d'index secondaires globaux dans DynamoDB
- Utilisation des transactions
- Accélération en mémoire avec DynamoDB Accelerator (DAX)
- Programmation avec DynamoDB et AWS SDKs

Générez du code d'infrastructure pour Amazon DynamoDB à l'aide de Console-to-Code

La Console-to-Code fonctionnalité Amazon Q Developer simplifie la gestion de l'infrastructure pour Amazon DynamoDB en transformant les étapes de création manuelle de tables en code d'automatisation reproductible. Cette fonctionnalité aide les développeurs à mettre à l'échelle efficacement la configuration des ressources de base de données dans leurs environnements. Pour plus d'informations, consultez Automatiser Services AWS avec Amazon Q Developer Console-to-Code.

Console-to-Code capture les configurations détaillées des tables DynamoDB, y compris les clés de partition, les clés de tri, les paramètres de débit provisionnés et les index secondaires, et les convertit en modèles précis. infrastructure-as-code Grâce à l'IA générative, cet outil garantit que le code généré maintient la compatibilité des paramètres établie au cours du flux de travail de la console.

Les développeurs peuvent générer du code d'infrastructure DynamoDB dans plusieurs formats, tels que :

- AWS Cloud Development Kit (AWS CDK) TypeScript en Python et Java
- CloudFormation en YAML ou JSON

Cette approche permet aux équipes de :

- normaliser la gestion des ressources de base de données ;
- mettre en œuvre une infrastructure contrôlée par version ;
- réduire les erreurs de configuration manuelle.

Console-to-Code pour Amazon DynamoDB est disponible dans toutes les régions AWS commerciales, fournissant une solution puissante pour transformer les configurations manuelles en code d'infrastructure automatisé et reproductible.

Comment ça marche

Lors de l'utilisation Console-to-Code avec DynamoDB, le processus implique généralement :

1. Prototypage dans la console : utilisez la console DynamoDB pour créer et configurer des ressources telles que des tables. Consultez Connect to Amazon DynamoDB pour plus d'informations.
2. Enregistrement des actions : Console-to-Code enregistre ces actions au fur et à mesure que vous les effectuez.
3. Génération de code : cette fonctionnalité utilise les capacités d'IA générative d'Amazon Q Developer pour transformer les actions de votre console en code réutilisable dans le format de votre choix.
4. Personnalisation du code : vous pouvez ensuite copier ou télécharger ce code et le personnaliser davantage en fonction de vos charges de travail de production.

Avantages de l'utilisation Console-to-Code avec DynamoDB

Automatisation simplifiée

Convertissez la création et la configuration manuelles de tables DynamoDB en code réutilisable d’un simple clic.

Bonnes pratiques

Le code généré suit les meilleures pratiques AWS guidées pour des déploiements fiables.

Pont entre la console et le code

Vous n'avez pas à choisir entre utiliser le AWS Management Console ou Infrastructure-as-Code (iAc). Vous pouvez utiliser les deux approches ensemble.

Développement accéléré

Démarrez rapidement avec un code d’automatisation qui peut être davantage personnalisé pour une utilisation en production.

Exemples de cas d’utilisation

- Création de tables DynamoDB avec des attributs, des clés et des paramètres de capacité spécifiques
- Configuration d’index secondaires globaux et d’index secondaires locaux
- Configuration de politiques d’autoscaling pour les tables DynamoDB
- Définition de configurations de sauvegarde et de restauration
- Création et gestion de DynamoDB Streams

Prise en main

Pour commencer à utiliser Console-to-Code DynamoDB :

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/DynamoDB/
2. Commencez à créer ou à modifier des ressources DynamoDB via l’interface de console.
3. Utilisez Console-to-Code cette fonctionnalité pour générer du code pour vos actions dans le format de votre choix.

4. Copiez ou téléchargez le code généré et personnalisez-le selon vos besoins spécifiques.

Pour plus d'informations et des instructions détaillées sur l'utilisation Console-to-Code, consultez Automating Services AWS with Amazon Q Developer Console-to-Code dans le guide de l'utilisateur Amazon Q Developer.

Fonctionnement d’Amazon DynamoDB

Les sections suivantes présentent les composants du service Amazon DynamoDB et leurs interactions.

Rubriques

- Aide-mémoire pour DynamoDB
- Composants de base d’Amazon DynamoDB
- API DynamoDB
- Types de données et règles de dénomination pris en charge dans Amazon DynamoDB
- Classes de tables DynamoDB
- Partitions et distribution des données dans DynamoDB
- Découvrez comment passer de SQL à NoSQL
- Ressources et outils de formation Amazon DynamoDB

Aide-mémoire pour DynamoDB

Ce aide-mémoire fournit une référence rapide pour travailler avec Amazon DynamoDB et ses différents outils.AWS SDKs

Configuration initiale du

1. Inscrivez-vous pour AWS.
2. Obtenez une clé d’accès AWS pour accéder à DynamoDB par programmation.
3. Configurez vos informations d’identification DynamoDB.

Voir aussi :

- Configuration de DynamoDB (service web)
- Mise en route avec DynamoDB
- Présentation de base des principaux composants

SDK ou CLI

Choisissez votre SDK préféré ou configurez l'AWS CLI.

> **Note**
>
> Lorsque vous utilisez Windows, une barre oblique inverse (\) qui ne figure pas dans un devis est considérée comme un retour de transport.AWS CLI De plus, vous devez éviter les guillemets et les accolades à l'intérieur d'autres guillemets. À titre d'exemple, reportez-vous à l'onglet Windows dans « Create a table » (Créer une table) dans la section suivante.

Voir aussi :

- AWS CLI avec DynamoDB
- Mise en route avec DynamoDB : étape 2

Actions de base

Cette section fournit le code pour les tâches de base de DynamoDB. Pour plus d'informations sur ces tâches, consultez Getting started with DynamoDB and the.AWS SDKs

Création d'une table

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows

```
aws dynamodb create-table ^
```

```
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

Écrire un élément dans une table

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

Lire un élément à partir d'une table

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Supprimer un élément d'une table

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Interroger une table

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Supprimer une table

```
aws dynamodb delete-table --table-name Music
```

Répertorier le nom des tables

```
aws dynamodb list-tables
```

Règles de dénomination

- Tous les noms doivent être codés en UTF-8 et sont sensibles à la casse.

- Les noms de table et les noms d'index doivent être compris entre 3 et 255 caractères, et peuvent contenir uniquement les caractères suivants :
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_`(soulignement)
  - `-`(trait d'union)
  - `.`(point)
- Un nom d'attribut doit compter au moins un caractère, mais ne peut pas avoir une taille supérieure à 64 Ko.

Pour plus d'informations, consultez les règles de dénomination.

Principes de base des quotas de service

Unités de lecture et d'écriture

- RCU (Read capacity unit) : une unité de capacité de lecture équivaut à une lecture fortement cohérente par seconde (ou à deux lectures éventuellement cohérentes par seconde) d'éléments dont la taille peut atteindre 4 Ko.
- WCU (Write capacity unit) : Une unité de capacité d'écriture équivaut à une écriture par seconde d'éléments dont la taille peut atteindre 1 Ko.

Limites liées aux tables

- Taille de la table : il n'existe pas de limite concrète de la taille d'une table. Les tables sont sans contraintes en ce qui concerne le nombre d'éléments ou le nombre d'octets.
- Nombre de tables — Pour chaque AWS compte, il existe un quota initial de 2 500 tables par AWS région.
- Limite de taille de page pour les requêtes et les analyses : la limite est de 1 Mo par page, par requête ou analyse. Si les paramètres de votre requête ou votre opération d'analyse sur une table génèrent plus de 1 Mo de données, DynamoDB renvoie les éléments correspondants initiaux. Il renvoie également une propriété `LastEvaluatedKey` que vous pouvez utiliser dans une nouvelle demande pour lire la page suivante.

Index

- Index secondaires locaux (LSIs) — Vous pouvez définir un maximum de cinq index secondaires locaux. LSIs sont principalement utiles lorsqu'un index doit avoir une forte cohérence avec la table de base.
- Index secondaires globaux (GSIs) : le quota par défaut est de 20 index secondaires globaux par table.
- Attributs d'index secondaire projeté par table : vous pouvez projeter jusqu'à un total de 100 attributs dans l'ensemble des index secondaires locaux et globaux d'une table. Cette possibilité ne s'applique qu'aux attributs projetés spécifiés par l'utilisateur.

Clé de partition

- La longueur minimale d'une valeur de clé de partition est 1 octet. La longueur maximale est de 2 048 octets.
- Il n'existe pas de limite pratique quant au nombre de valeurs de clé de partition distinctes, pour les tables ou pour les index secondaires.
- La longueur minimale d'une valeur de clé de tri est 1 octet. La longueur maximale est de 1 024 octets.
- En général, il n'existe pas de limite pratique sur le nombre de valeurs de clé de tri distinctes par valeur de clé de partition. L'exception concerne les tables avec index secondaires.

Pour plus d'informations sur les index secondaires, ainsi que sur la conception des clés de partition et des clés de tri, consultez la section Bonnes pratiques.

Limites applicables aux types de données couramment utilisés

- Chaîne : la longueur d'une chaîne est limitée par la taille maximum de l'élément, qui de 400 Ko. Les chaînes sont Unicode avec codage binaire UTF-8.
- Nombre : un nombre peut avoir jusqu'à 38 chiffres de précision et peut être positif, négatif ou nul.
- Binaire : la longueur d'un élément de type binaire est limitée par la taille maximum de l'élément, qui de 400 Ko. Les applications qui fonctionnent avec des attributs de type binaire doivent encoder les données au format base 64 avant de les envoyer à DynamoDB.

Pour afficher la liste complète des types de données pris en charge, consultez Types de données. Pour de plus amples informations, veuillez également consulter Quotas de service.

Éléments, attributs et paramètres d'expression

La taille maximum d'un élément dans DynamoDB est de 400 Ko, ce qui comprend la longueur binaire du nom d'attribut (longueur UTF-8) et les longueurs binaires de valeur d'attribut (longueur UTF-8). Le nom d'attribut est comptabilisé parmi la limite de taille.

Il n'existe aucune limite au nombre de valeurs dans un élément de type List (liste), Map (mappage) ou Set (ensemble), pour autant que la taille de l'élément ne dépasse pas la talle limite de 400 Ko.

Pour les paramètres d'expression, la longueur maximale d'une chaîne d'expression est de 4 Ko.

Pour plus d'informations sur la taille des éléments, les attributs et les paramètres d'expression, consultez la section Quotas de service.

En savoir plus

- Sécurité
- Surveillance et journalisation
- Utilisation des flux
- Sauvegardes et Point-in-time restauration
- Intégration à d'autres AWS services
- Référence d'API
- Centre d'architecture : bonnes pratiques en matière de bases de données
- Didacticiels vidéo
- Forum DynamoDB

Composants de base d'Amazon DynamoDB

Dans DynamoDB, les tables, les éléments et les attributs sont les principaux composants que vous utilisez. Une table est une collection d'éléments et chaque élément est une collection d'attributs. DynamoDB utilise des clés primaires afin d'identifier de façon unique chaque élément d'une table. Vous pouvez utiliser DynamoDB Streams pour récupérer les événements de modification des données dans des tables DynamoDB.

Il existe des limites dans DynamoDB. Pour de plus amples informations, veuillez consulter Quotas dans Amazon DynamoDB.

La vidéo suivante présente des tables, des éléments et des attributs.

Tables, éléments et attributs

Tables, éléments et attributs

People

Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

Voici les composants DynamoDB de base :

- Tables – Comme d'autres systèmes de base de données, DynamoDB stocke les données dans des tables. Une table est un ensemble de données. Par exemple, consultez l'exemple de table appelée People que vous pouvez utiliser pour stocker les informations sur vos amis, votre famille ou toute autre personne de votre choix. Vous pouvez également avoir une table Cars pour stocker les informations sur les véhicules que les personnes conduisent.
- Eléments – Chaque table contient zéro ou plusieurs éléments. Un élément est un groupe d'attributs identifiable de façon unique parmi tous les autres éléments. Dans une table People, chaque élément représente une personne. Pour une table Cars, chaque élément représente un véhicule. Les éléments dans DynamoDB sont similaires à maints égards aux lignes, registres ou tuples dans d'autres systèmes de base de données. Dans DynamoDB, il n'existe pas de limite au nombre d'éléments que vous pouvez stocker dans une table.
- Attributs – Chaque élément se compose d'un ou de plusieurs attributs. Un attribut est un élément de donnée fondamental, qui n'a pas besoin d'être décomposé plus avant. Par exemple, un élément d'une table People contient des attributs appelés PersonId LastName, FirstName,, etc. Pour une table Department, un élément peut avoir des attributs tels que DepartmentID, Name, Manager, etc.

Les attributs dans DynamoDB sont similaires à maints égards à des champs ou colonnes dans d'autres systèmes de base de données.

Le schéma suivant montre une table nommée Personnes avec certains exemples d'éléments et d'attributs.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Notez ce qui suit à propos de la table Personnes :

- Chaque élément de la table possède un identifiant unique, ou clé primaire, qui distingue l'élément de tous les autres éléments de la table. Dans la table Personnes, la clé primaire se compose d'un seul attribut (IDPersonne).
- En dehors de la clé primaire, la table Personnes est sans schéma, ce qui signifie que ni les attributs ni leurs types de données ne doivent être définis au préalable. Chaque élément peut avoir ses propres attributs distincts.
- La plupart des attributs sont scalaires, ce qui signifie qu'ils ne peuvent avoir qu'une seule valeur. Les chaînes et les nombres sont des exemples courants de scalaires.
- Certains éléments ont un attribut imbriqué (Address). DynamoDB prend en charge les attributs imbriqués jusqu'à 32 niveaux de profondeur.

Ce qui suit est un autre exemple de table nommée Musique que vous pouvez utiliser pour effectuer le suivi de votre collection musicale.

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
```

```
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Notez ce qui suit à propos de la table Musique :

- La clé primaire de Music se compose de deux attributs (Artiste et SongTitle). Chaque élément de la table doit avoir ces deux attributs. La combinaison de Artist et C SongTitledistingue chaque élément du tableau de tous les autres.
- En dehors de la clé primaire, la table Musique est sans schéma, ce qui signifie que ni les attributs ni leurs types de données ne doivent être définis au préalable. Chaque élément peut avoir ses propres attributs distincts.
- L'un des éléments possède un attribut imbriqué (PromotionInfo), qui contient d'autres attributs imbriqués. DynamoDB prend en charge les attributs imbriqués jusqu'à 32 niveaux de profondeur.

Pour de plus amples informations, veuillez consulter Utilisation de tables et de données dans DynamoDB.

Clé primaire

Lorsque vous créez une table, en plus du nom de la table, vous devez spécifier la clé primaire de la table. La clé primaire identifie de manière unique chaque élément de la table, afin qu'aucun deux éléments n'ait la même clé.

DynamoDB prend en charge deux genres de clés primaires :

- Clé de partition – Clé primaire simple, composée d'un attribut nommé clé de partition.

  DynamoDB utilise la valeur de la clé de partition comme entrée pour une fonction de hachage interne. La sortie de la fonction de hachage détermine la partition (stockage physique interne de DynamoDB) dans laquelle l'élément sera stocké.

  Dans une table comportant une seule clé de partition, deux éléments d'une table ne peuvent pas avoir la même valeur de clé de partition.

  La table Personnes décrite dans Tables, éléments et attributs est un exemple de table avec une clé primaire simple (IDPersonne). Vous pouvez accéder directement à n'importe quel élément du tableau Personnes en fournissant la PersonIdvaleur de cet élément.
- Clé de partition et clé de tri – Appelée clé primaire composite, ce type de clé se compose de deux attributs. Le premier attribut est la clé de partition et le second attribut est la clé de tri.

  DynamoDB utilise la valeur de clé de partition comme entrée pour une fonction de hachage interne. La sortie de la fonction de hachage détermine la partition (stockage physique interne de DynamoDB) dans laquelle l'élément sera stocké. Tous les éléments avec la même valeur de clé de partition sont stockés ensemble, par ordre de valeur de la clé de tri.

  Dans une table comportant une clé de partition et une clé de tri, plusieurs éléments d'une table peuvent avoir la même valeur clé de partition. Toutefois, ces éléments doivent avoir des valeurs clé de tri différentes.

  La table Music décrite dans Tables, éléments et attributs est un exemple de table dotée d'une clé primaire composite (Artist et SongTitle). Vous pouvez accéder directement à n'importe quel élément du tableau Musique, si vous indiquez l'artiste et SongTitleles valeurs de cet élément.

  Une clé primaire composite vous offre plus de flexibilité lors de l'interrogation des données. Par exemple, si vous fournissez uniquement la valeur pour Artiste, DynamoDB extrait tous les morceaux de cet artiste. Pour récupérer uniquement un sous-ensemble de chansons d'un artiste

en particulier, vous pouvez fournir une valeur pour Artist ainsi qu'une plage de valeurs pour SongTitle.

> **Note**
>
> La clé de partition d'un élément est également appelée attribut de hachage. L'expression attribut de hachage dérive de l'utilisation d'une fonction de hachage interne dans DynamoDB qui répartit uniformément les éléments de données entre les partitions, en fonction de leurs valeurs de clé de partition.
>
> La clé de tri d'un élément est également appelée attribut de plage. L'expression attribut de plage dérive de la façon dont DynamoDB stocke des éléments ayant la même clé de partition à proximité physique les uns des autres, dans un ordre trié sur la valeur de clé de tri.

Chaque attribut de clé primaire doit être un scalaire (ce qui signifie qu'il ne peut contenir qu'une seule valeur). Les seuls types de données autorisés pour les attributs de clé primaires sont string, number ou binary. Il n'y a aucune restriction semblable pour les autres attributs non-clés.

Index secondaires

Vous pouvez créer un ou plusieurs index secondaires sur une table. Un index secondaire vous permet d'interroger les données de la table à l'aide d'une clé alternative, en plus des requêtes sur la clé primaire. DynamoDB n'exige pas l'utilisation d'index, mais ceux-ci apportent à vos applications davantage de flexibilité pour l'interrogation des données. Une fois que vous avez créé un index secondaire sur une table, vous pouvez lire les données à partir de l'index de la même façon que vous le feriez à partir de la table.

DynamoDB prend en charge deux types d'index :

- Index secondaire global – Index avec une clé de partition et une clé de tri pouvant différer de celles de la table. Les valeurs de clé primaire dans les index secondaires globaux ne doit pas nécessairement être uniques.
- Index secondaire local – Index avec la même clé de partition que la table, mais une clé de tri différente.

Dans DynamoDB, les index secondaires globaux GSIs () sont des index qui s'étendent sur l'ensemble de la table, ce qui vous permet d'effectuer des requêtes sur toutes les clés de partition. Les index

secondaires locaux (LSIs) sont des index qui ont la même clé de partition que la table de base mais une clé de tri différente.

Chaque table dans DynamoDB a un quota de 20 index secondaires globaux (quota par défaut) et de 5 index secondaires locaux.

Dans l'exemple de table Music présenté précédemment, vous pouvez interroger des éléments de données par artiste (clé de partition) ou par artiste et SongTitle(clé de partition et clé de tri). Et si vous vouliez également interroger les données par genre et AlbumTitle? Pour ce faire, vous pouvez créer un index sur Genre AlbumTitle, puis interroger l'index de la même manière que vous interrogeriez la table Music.

Le schéma suivant montre l'exemple de table musicale, avec un nouvel index appelé GenreAlbumTitle. Dans l'index, Genre est la clé de partition et AlbumTitlela clé de tri.

Table Music	GenreAlbumTitle

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Hey Now",
	    "Artist": "No One You Know",
	    "SongTitle": "My Dog Spot"
	}</pre>

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The
 Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Somewhat Famous",
	    "Artist": "No One You Know",
	    "SongTitle": "Somewhere Down The
	 Road"
	}</pre>

Table Music	GenreAlbumTitle

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": {
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        },
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Still In Love"
	}</pre>

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 0.99,
    "Genre": "Rock"
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Look Out, World"
	}</pre>

Notez ce qui suit à propos de l'GenreAlbumTitleindex :

- Chaque index appartient à une table, qui est appelée la table de base de l'index. Dans l'exemple précédent, Music est la table de base de l'GenreAlbumTitleindex.
- DynamoDB tient à jour les index automatiquement. Lorsque vous ajoutez, mettez à jour ou supprimez un élément dans la table de base, DynamoDB ajoute, met à jour ou supprime l'élément correspondant dans les index appartenant à cette table.
- Lorsque vous créez un index, vous spécifiez quels attributs seront copiés, ou projetés, depuis la table de base vers l'index. Au minimum, DynamoDB projette les attributs de clé de la table de base dans l'index. Tel est le cas avec `GenreAlbumTitle`, où seuls les attributs clés de la table `Music` sont projetés dans l'index.

Vous pouvez consulter l'GenreAlbumTitleindex pour trouver tous les albums d'un genre donné (par exemple, tous les albums Rock). Vous pouvez également interroger l'index pour rechercher tous les albums dans un genre particulier avec certains titres d'album (par exemple, tous les albums Country dont les titres commencent par la lettre H).

Pour de plus amples informations, veuillez consulter Amélioration de l'accès aux données avec les index secondaires dans DynamoDB.

DynamoDB Streams

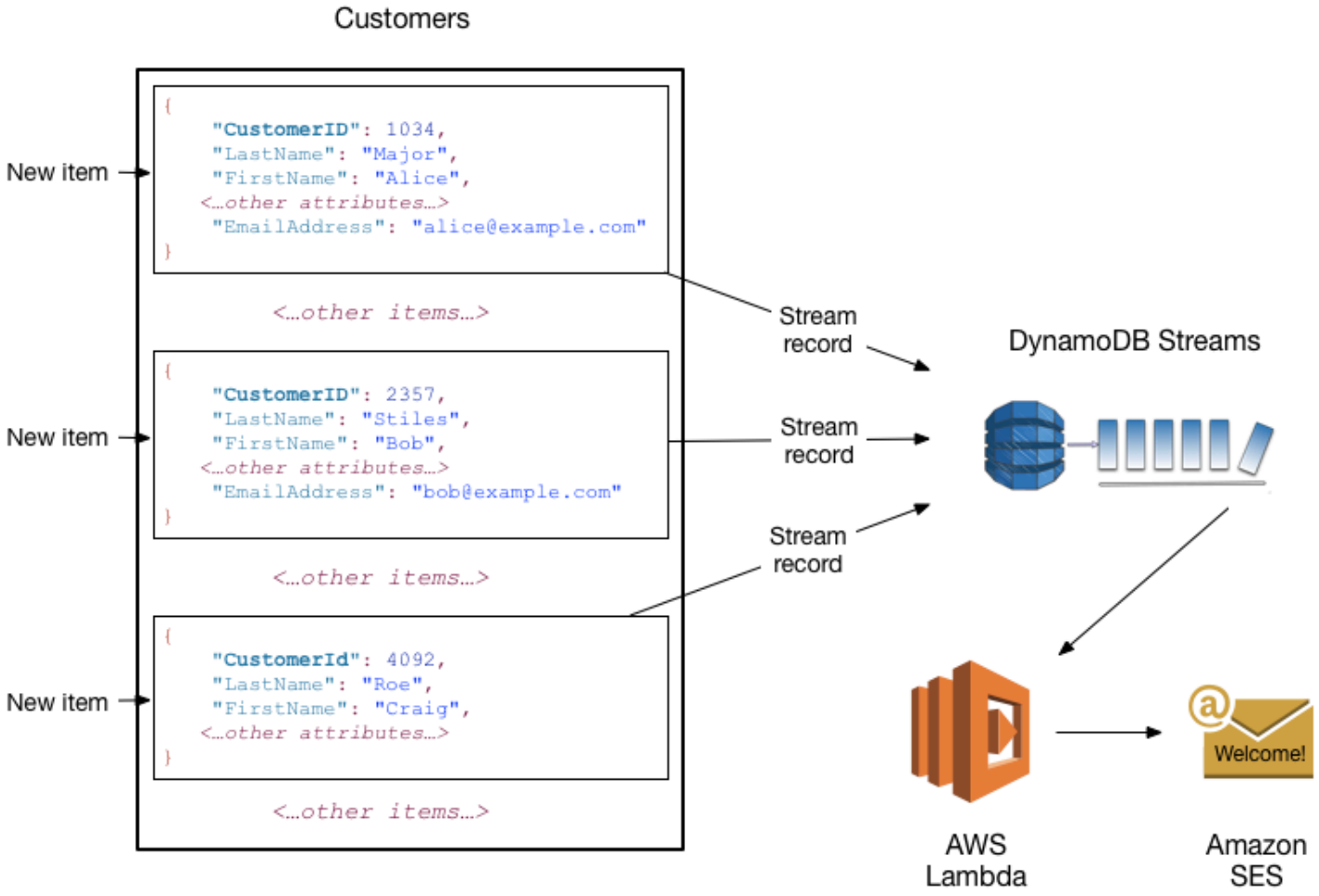
DynamoDB Streams est une fonction facultative qui récupère les événements de modification de données dans des tables DynamoDB. Les données sur ces événements apparaissent dans le flux de données presque en temps réel et dans l'ordre où les événements se sont produits.

Chaque événement est représenté par un enregistrement de flux. Si vous activez un flux sur une table, DynamoDB Streams écrit un registre de flux chaque fois que l'un des événements suivants se produit :

- Un nouvel élément est ajouté à la table : le flux récupère une image de la totalité de l'élément, y compris l'ensemble de ses attributs.
- Un élément est mis à jour : le flux capture l'image « avant » et « après » des attributs qui ont été modifiés dans l'élément.
- Un élément est supprimé de la table : Le flux saisit une image de la totalité de l'élément avant qu'il n'ait été supprimé.

Chaque enregistrement de flux contient aussi le nom de la table, l'horodatage de l'événement et autres métadonnées. Les enregistrements de flux ont une durée de vie de 24 heures ; passé ce délai, ils sont automatiquement supprimés du flux.

Vous pouvez utiliser DynamoDB Streams conjointement AWS Lambda pour créer un déclencheur, c'est-à-dire un code qui s'exécute automatiquement chaque fois qu'un événement intéressant apparaît dans un flux. Par exemple, imaginons une table Clients qui contient les informations client d'une entreprise. Supposons que vous souhaitiez envoyer un e-mail de bienvenue à chaque nouveau client. Vous pouvez activer un flux sur cette table, puis associer le flux à une fonction Lambda. La fonction Lambda s'exécute chaque fois qu'un nouveau registre de flux apparaît, mais traite uniquement les nouveaux éléments ajoutés à la table Customers. Pour tout élément ayant un attribut `EmailAddress`, la fonction Lambda appelle Amazon Simple Email Service (Amazon SES) pour envoyer un e-mail à cette adresse.



> **Note**
>
> Dans cet exemple, le dernier client, Craig Roe, ne reçoit pas d'e-mail, car il n'a pas d'attribut `EmailAddress`.

Outre les déclencheurs, DynamoDB Streams propose des solutions puissantes telles que la réplication de données au sein des AWS régions et entre celles-ci, les vues matérialisées des données dans des tables DynamoDB, l'analyse des données à l'aide de vues matérialisées Kinesis, et bien plus encore.

Pour de plus amples informations, veuillez consulter Modifier la récupération de données pour DynamoDB Streams.

API DynamoDB

Pour opérer avec Amazon DynamoDB, votre application doit utiliser quelques opérations d'API simples. Voici un résumé de ces opérations, classées par catégorie.

> **Note**
>
> Pour obtenir la liste complète des opérations d'API, consultez la Référence API Amazon DynamoDB.

Rubriques

- Plan de contrôle
- Plan de données
- DynamoDB Streams
- Transactions

Plan de contrôle

Les opérations de plan de contrôle vous permettent de créer et gérer des tables DynamoDB. Elles vous permettent également d'utiliser les index, les flux et autres objets qui dépendent des tables.

- `CreateTable` – Crée une table. Vous pouvez créer un ou plusieurs index secondaires, puis activer DynamoDB Streams pour la table.
- `DescribeTable` – Retourne des informations sur une table, telles que son schéma de clé primaire, ses paramètres de débit et ses informations d'index.
- `ListTables` – Renvoie les noms de toutes vos tables dans une liste.
- `UpdateTable` – Modifie les paramètres d'une table ou de ses index, crée ou supprime de nouveaux index sur une table, ou modifie les paramètres DynamoDB Streams d'une table.
- `DeleteTable` – Supprime une table et tous ses objets dépendants de DynamoDB.

Plan de données

Les opérations de plan de données vous permettent d'exécuter les opérations de création, lecture, mise à jour et suppression (également appelées opérations CRUD) sur les données d'une table. Certaines opérations de plan de données vous permettent également de lire les données d'un index secondaire.

Vous pouvez utiliserPartiQL – Langage de requête compatible SQL pour Amazon DynamoDB, pour effectuer ces opérations CRUD, ou vous pouvez utiliser le CRUD classique de DynamoDB APIs qui sépare chaque opération en un appel d'API distinct.

PartiQL – Langage de requête compatible SQL

- `ExecuteStatement` – Lit plusieurs éléments d'une table. Vous pouvez également écrire ou mettre à jour un élément unique à partir d'une table. Lors de l'écriture ou de la mise à jour d'un élément unique, vous devez spécifier les attributs de clé primaire.
- `BatchExecuteStatement` – Ecrit, met à jour ou lit plusieurs éléments d'une table. Cette solution est plus efficace que l'opération `ExecuteStatement` parce que votre application n'a besoin que d'un seul aller-retour réseau pour lire les éléments.

classique APIs

Création de données

- `PutItem` – Ecrit un élément unique dans une table. Vous devez spécifier les attributs de clé primaire, mais vous n'avez pas à spécifier d'autres attributs.

- `BatchWriteItem` – Ecrit jusqu'à 25 éléments dans une table. Cette solution est plus efficace que l'appel répété de `PutItem` parce que votre application a uniquement besoin d'un seul aller-retour réseau pour écrire les éléments.

Lecture de données

- `GetItem` – Extrait un élément unique d'une table. Vous devez spécifier la clé primaire de l'élément que vous voulez. Vous pouvez récupérer l'élément entier ou juste un sous-ensemble de ses attributs.
- `BatchGetItem` – Extrait jusqu'à 100 éléments d'une ou de plusieurs tables. Cette solution est plus efficace que l'appel répété de `GetItem` parce que votre application a uniquement besoin d'un seul aller-retour réseau pour lire les éléments.
- `Query` – Extrait tous les éléments ayant une clé de partition spécifique. Vous devez spécifier la valeur de la clé de partition. Vous pouvez récupérer des éléments entiers, ou juste un sous-ensemble de leurs attributs. Le cas échéant, vous pouvez appliquer une condition aux valeurs de clé de tri, afin de récupérer uniquement un sous-ensemble des données ayant la même clé de partition. Vous pouvez utiliser cette opération sur une table, à condition que la table ait une clé de partition et une clé de tri. Vous pouvez également utiliser cette opération sur un index, à condition que l'index ait une clé de partition et une clé de tri.
- `Scan` – Extrait tous les éléments de la table ou de l'index spécifiés. Vous pouvez récupérer des éléments entiers, ou juste un sous-ensemble de leurs attributs. Le cas échéant, vous pouvez appliquer une condition de filtre pour renvoyer uniquement les valeurs qui vous intéressent et ignorer les autres.

Mise à jour de données

- `UpdateItem` – Modifie un ou plusieurs attributs dans un élément. Vous devez spécifier la clé primaire de l'élément que vous voulez modifier. Vous pouvez ajouter de nouveaux attributs et modifier ou supprimer des attributs existants. Vous pouvez également effectuer des mises à jour conditionnelles, afin que la mise à jour ne réussisse que lorsqu'une condition définie par l'utilisateur est remplie. Le cas échéant, vous pouvez mettre en place un compteur atomique, qui augmente ou diminue un attribut numérique sans interférer avec d'autres demandes d'écriture.

Suppression des données

- `DeleteItem` – Supprime un élément unique d'une table. Vous devez spécifier la clé primaire de l'élément que vous voulez supprimer.
- `BatchWriteItem` – Supprime jusqu'à 25 éléments d'une ou de plusieurs tables. Cette solution est plus efficace que l'appel répété de `DeleteItem` parce que votre application a uniquement besoin d'un seul aller-retour réseau pour supprimer les éléments.

> **Note**
>
> Vous pouvez utiliser `BatchWriteItem` pour créer des données et supprimer des données.

DynamoDB Streams

Les opérations DynamoDB Streams vous permettent d'activer ou de désactiver un flux sur une table, et d'autoriser l'accès aux registres de modification de données contenus dans un flux.

- `ListStreams` – Retourne la liste de tous vos flux, ou simplement le flux d'une table spécifique.
- `DescribeStream` – Retourne des informations sur un flux, telles que son Amazon Resource Name (ARN) et l'emplacement où votre application peut commencer à lire les tout premiers registres de flux.
- `GetShardIterator` – Retourne un itérateur de partition, c'est-à-dire une structure de données que votre application utilise pour extraire les registres du flux.
- `GetRecords` – Récupère un ou plusieurs registres de flux à l'aide d'un itérateur de partition donné.

Transactions

Les transactions offrent atomicité, cohérence, isolation et durabilité (ACID), ce qui permet de maintenir plus facilement l'exactitude des données dans vos applications.

Vous pouvez utiliserPartiQL – Langage de requête compatible SQL pour Amazon DynamoDB, pour effectuer des opérations transactionnelles, ou vous pouvez utiliser le CRUD classique de DynamoDB APIs qui sépare chaque opération en un appel d'API distinct.

PartiQL – Langage de requête compatible SQL

- `ExecuteTransaction`— Une opération par lots qui permet d'effectuer des opérations CRUD sur plusieurs éléments à la fois dans et entre les tables avec un all-or-nothing résultat garanti.

classique APIs

- `TransactWriteItems`— Une opération par lots qui permet d'`PutUpdate`effectuer des `Delete` opérations sur plusieurs éléments à la fois dans et entre les tables avec un all-or-nothing résultat garanti.
- `TransactGetItems` – Opération par lot qui permet d'effectuer des opérations `Get` pour extraire plusieurs éléments d'une ou de plusieurs tables.

Types de données et règles de dénomination pris en charge dans Amazon DynamoDB

Cette section décrit les règles de dénomination dans Amazon DynamoDB, ainsi les différents types de données que DynamoDB prend en charge. Des limites s'appliquent aux types de données. Pour de plus amples informations, veuillez consulter Types de données.

Rubriques

- Règles de dénomination
- Types de données
- Descripteurs de type de données

Règles de dénomination

Les tables, attributs et autres objets dans DynamoDB doivent avoir des noms. Les noms doivent être éloquents et concis. Par exemple, des noms tels que Produits, Livres et Auteurs sont explicites.

Voici les règles de dénomination pour DynamoDB.

- Tous les noms doivent être codés à l'aide d'UTF-8 et sont sensibles à la casse.
- Les noms de table et les noms d'index doivent être compris entre 3 et 255 caractères, et peuvent contenir uniquement les caractères suivants :

- `a-z`
- `A-Z`
- `0-9`
- `_` (soulignement)
- `-` (tiret)
- `.` (point)

- Un nom d'attribut doit compter au moins un caractère, mais sa taille doit être inférieure à 64 Ko. Une bonne pratique consiste à utiliser des noms d'attribut aussi courts que possible. Cela permet de réduire le nombre d'unités de demande de lecture consommées, car les noms d'attributs sont inclus dans la mesure du stockage et de l'utilisation de débit.

  Les éléments suivants sont les exceptions. Ces noms d'attribut ne doivent pas dépasser 255 caractères :

  - Noms de clés de partition d'index secondaire
  - Noms de clés de tri d'index secondaire
  - Noms d'attributs projetés spécifiés par l'utilisateur (applicables uniquement aux index secondaires locaux)

Mots réservés et caractères spéciaux

DynamoDB comprend une liste de mots réservés et de caractères spéciaux. Pour obtenir la liste complète, consultez Mots réservés dans DynamoDB. Les caractères suivants ont également une signification spéciale dans DynamoDB : # (dièse) et : (deux points).

Même si DynamoDB vous permet d'utiliser ces mots réservés et ces caractères spéciaux pour les noms, nous vous recommandons d'éviter de le faire, car vous devez définir les variables d'espace réservé chaque fois que vous utilisez ces noms dans une expression. Pour de plus amples informations, veuillez consulter Noms d'attributs d'expression (alias) dans DynamoDB.

Types de données

DynamoDB prend en charge différents types de données pour les attributs au sein d'une table. Ils peuvent être classés comme suit :

- Types scalar (scalaire) – Un type scalar peut représenter exactement une valeur. Les types scalar sont les suivants : number, string, binary, Boolean et null.

- Types document – Un type document peut représenter une structure complexe constituée d'attributs imbriqués, comme vous pouvez en trouver dans un document JSON. Les types document sont les suivants : list et map.
- Types set (ensemble) – Un type set peut représenter plusieurs valeurs scalaires. Les types set sont les suivants : string set, number set et binary set.

Lorsque vous créez une table ou un index secondaire, vous devez spécifier les noms et les types de données de chaque attribut de clé primaire (clé de partition et clé de tri). En outre, chaque attribut de clé primaire doit être défini en tant que type string, number ou binary (chaîne, nombre ou binaire).

DynamoDB est une base de données NoSQL et est sans schéma. Cela signifie que, en dehors des attributs de clé primaire, vous n'avez pas besoin de définir d'attributs ou de types de données lorsque vous créez des tables. En comparaison, vous devez, pour les bases de données relationnelles, définir les noms et les types de données de chaque colonne lorsque vous créez une table.

Les descriptions suivantes concernent chaque type de données, ainsi que des exemples au format JSON.

Types scalar

Les types scalar sont les suivants : number, string, binary, Boolean et null.

Number

Les nombres peuvent être positifs, négatifs ou nuls. La précision maximum des nombres est de 38 chiffres. Tout dépassement entraîne une exception. Si vous avez besoin d'une précision supérieure à 38 chiffres, vous pouvez utiliser des chaînes.

- Plage positive : 1E-130 à 9,99999999999999999999999999999999999999E+125
- Plage négative : -9,99999999999999999999999999999999999999E+125 à -1E-130

Dans DynamoDB, les nombres sont représentés en tant que longueur variable. Les zéros de début et de fin sont tronqués.

Tous les nombres sont envoyés à DynamoDB via le réseau sous forme de chaînes afin d'optimiser la compatibilité entre les langages et les bibliothèques. Toutefois, DynamoDB les traite en tant qu'attributs de type number (nombre) pour les opérations mathématiques.

Vous pouvez utiliser le type de données number pour représenter une date ou un horodatage. Une façon de procéder consiste à utiliser l'heure epoch, à savoir le nombre de secondes depuis le 1er janvier 1970 à 00:00:00 UTC. Par exemple, l'heure Posix `1437136300` correspond au 17 juillet 2015, 12:31:40 UTC.

Pour plus d'informations, consultez http://en.wikipedia. org/wiki/Unix_heure.

String

Les chaînes sont Unicode avec codage binaire UTF-8. La longueur minimale d'une chaîne peut être zéro si l'attribut n'est pas utilisé comme clé pour un index ou une table, et est contrainte par la limite de taille d'élément de 400 Ko dans DynamoDB.

Les contraintes supplémentaires suivantes s'appliquent aux attributs de clé primaire définis comme type string (chaîne) :

- Pour une clé primaire unique, la longueur maximale de la première valeur d'attribut (la clé de partition) est de 2 048 octets.
- Pour une clé primaire composite, la longueur maximale de la deuxième valeur d'attribut (la clé de tri) est de 1 024 octets.

DynamoDB collationne et compare les chaînes à l'aide des octets de l'encodage de chaîne UTF-8 sous-jacent. Par exemple, « a » (0x61) est supérieur à « A » (0x41) et « ¿ » (0xC2BF) est supérieur à « z » (0x7A).

Vous pouvez utiliser le type de données string pour représenter une date ou un horodatage. Une solution pour cela consiste à utiliser les chaînes ISO 8601, comme illustré dans ces exemples :

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Pour plus d'informations, consultez http://en.wikipedia. org/wiki/ISO_8601.

> **Note**
>
> Contrairement aux bases de données relationnelles classiques, DynamoDB ne prend pas en charge de manière native un type de données de date et d'heure. Il peut être utile de stocker

les données et les données temporelles sous forme de type de données numériques, en utilisant le temps d'époque Unix.

Binaire

Les attributs de type binary peuvent stocker n'importe quelle donnée binaire, telle que texte compressé, données chiffrées ou images. Chaque fois que DynamoDB compare des valeurs binaires, il traite chaque octet de données binaires comme non signé.

La longueur d'un binary attribute peut être zéro si l'attribut n'est pas utilisé comme clé pour un index ou une table, et est contrainte par la limite de taille d'élément de 400 Ko dans DynamoDB.

Si vous définissez un attribut de clé primaire comme attribut de type binary (binaire), les contraintes supplémentaires suivantes s'appliquent :

- Pour une clé primaire unique, la longueur maximale de la première valeur d'attribut (la clé de partition) est de 2 048 octets.
- Pour une clé primaire composite, la longueur maximale de la deuxième valeur d'attribut (la clé de tri) est de 1 024 octets.

Vos applications doivent encoder les valeurs binaires dans un format encodé en base 64 avant de les envoyer à DynamoDB. À la réception de ces valeurs, DynamoDB les décode dans un tableau d'octets non signés, et utilise ces informations comme longueur du binary attribute.

L'exemple suivant est un binary attribute, utilisant un texte codé en base64.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Booléen

Un attribut de type Boolean peut stocker `true` ou `false`.

Null

Null représente un attribut avec un état inconnu ou non défini.

Types document

Les types document sont les suivants : list et map. Ces types de données peuvent être imbriqués les uns dans les autres pour représenter des structures de données complexes jusqu'à 32 niveaux de profondeur.

Il n'existe aucune limite au nombre de valeurs dans un élément de type list (liste) ou map (mappage), pour autant que la taille de l'élément ne dépasse pas la limite de taille d'élément en vigueur dans DynamoDB (400 Ko).

Une valeur d'attribut peut être une chaîne vide ou une valeur binaire vide si l'attribut n'est pas utilisé pour une clé de table ou d'index. Une valeur d'attribut ne peut pas être un jeu vide (jeu de chaînes, jeu de chiffres ou jeu binaire). Cependant, les listes et les cartes vides sont autorisées. Les valeurs binaires et de chaînes vides sont autorisées dans les listes et les cartes. Pour de plus amples informations, veuillez consulter Attributes.

List

Un attribut de type list peut stocker une collection ordonnée de valeurs. Les types list sont entourés de crochets : `[ ... ]`

Une liste est similaire à un tableau JSON. Il n'y a aucune restriction sur les types de données qui peuvent être stockés dans un élément de type list et les éléments d'un élément de type list n'ont pas à être du même type.

L'exemple suivant montre une liste contenant deux chaînes et un nombre.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB vous permet d'utiliser des éléments individuels dans des listes, même si ces éléments sont profondément imbriqués. Pour de plus amples informations, veuillez consulter Utilisation d'expressions dans DynamoDB.

Map

Un attribut de type map peut stocker une collection non ordonnée de paires nom-valeur. Les types map sont placés entre accolades : `{ ... }`

Un type map est similaire à un objet JSON. Il n'y a aucune restriction sur les types de données qui peuvent être stockés dans un élément de type map et les éléments d'un élément de type map n'ont pas à être du même type.

Les éléments de type map sont idéaux pour stocker des documents JSON dans DynamoDB. L'exemple suivant montre un élément de type map qui contient une chaîne, un nombre et une liste imbriquée contenant un autre élément de type map.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB vous permet d'utiliser des éléments individuels dans des mappages, même si ces éléments sont profondément imbriqués. Pour de plus amples informations, veuillez consulter Utilisation d'expressions dans DynamoDB.

Sets

DynamoDB prend en charge des types représentant des ensembles de valeurs de type string (chaîne), number (nombre) ou binary (binaire). Tous les éléments au sein d'un ensemble doivent être du même type. Par exemple, un Number Set peut contenir uniquement des nombres, et un String Set peut contenir uniquement des chaînes.

Il n'existe aucune limite au nombre de valeurs dans un élément de type set (ensemble), pour autant que la taille de l'élément ne dépasse pas la limite de taille d'élément en vigueur dans DynamoDB (400 Ko).

Chaque valeur au sein d'un ensemble doit être unique. L'ordre des valeurs d'un ensemble n'est pas conservé. Par conséquent, vos applications ne doivent pas reposer sur un ordre particulier des éléments au sein de l'ensemble. DynamoDB ne prend pas en charge les ensembles vides. Cependant, des valeurs de type string (chaîne) et binary (binaire) vides sont autorisées dans un ensemble.

L'exemple suivant illustre un ensemble de chaînes, un ensemble de nombres et un ensemble de données binaires :

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Descripteurs de type de données

Le protocole d'API DynamoDB de bas niveau utilise des descripteurs de type de données comme jetons qui indiquent à DynamoDB comment interpréter chaque attribut.

La liste suivante est la liste complète des descripteurs de type de données DynamoDB :

- **`S`** – String (chaîne)
- **`N`** – Number (nombre)
- **`B`** – Binary (binaire)
- **`BOOL`** – Boolean (booléen)
- **`NULL`** – Null
- **`M`** – Map (mappage)
- **`L`** – List (liste)
- **`SS`** – String Set (ensemble de chaînes)
- **`NS`** – Number Set (ensemble de nombres)
- **`BS`** – Binary Set (ensemble de binaires)

Classes de tables DynamoDB

DynamoDB propose deux classes de tables conçues pour vous aider à optimiser vos coûts. La classe de tables DynamoDB Standard est la classe par défaut. Elle est recommandée pour la

grande majorité des charges de travail. La classe de tables DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) est optimisée pour les tables où le stockage est le coût dominant. Par exemple, les tables qui stockent des données rarement consultées, telles que les journaux d'applications, les anciennes publications sur les réseaux sociaux, l'historique des commandes d'e-commerce et les exploits de jeux passés, sont de bons candidats pour la classe de tables Standard-IA. Pour plus de détails sur la tarification, consultez Tarification Amazon DynamoDB.

Chaque table DynamoDB est associée à une classe de tables (DynamoDB Standard par défaut). Tous les index secondaires associés à la table utilisent la même classe de table. Chaque classe de tables offre une tarification différente pour le stockage des données ainsi que pour les demandes de lecture et d'écriture. Vous pouvez sélectionner la classe de tables la plus rentable pour votre table en fonction de ses modèles de stockage et d'utilisation du débit.

Le choix d'une classe de table n'est pas permanent. Vous pouvez modifier ce paramètre à l'aide de la AWS Management ConsoleAWS CLI ou du SDK.AWS DynamoDB prend également en charge la gestion de votre classe de table àAWS CloudFormation l'aide de tables à région unique et de tables globales. Pour en savoir plus sur la sélection de votre classe de tables, consultez Points à prendre en considération lors du choix d'une classe de tables dans DynamoDB.

Partitions et distribution des données dans DynamoDB

Amazon DynamoDB stocke les données dans des partitions. Une partition est une allocation de stockage pour une table, soutenue par des disques SSD (SSDs) et répliquée automatiquement sur plusieurs zones de disponibilité au sein d'une AWS région. La gestion des partitions étant entièrement effectuée par DynamoDB, vous n'avez jamais besoin de vous en occuper.

Lorsque vous créez une table, l'état initial de la table est `CREATING`. Au cours de cette phase, DynamoDB alloue à la table des partitions en nombre suffisant pour qu'elle puisse gérer vos exigences de débit approvisionné. Vous pouvez commencer à écrire et à lire les données de table une fois que l'état de la table est devenu `ACTIVE`.

DynamoDB alloue des partitions supplémentaires à une table dans les cas suivants :

- Si vous augmentez les paramètres de débit alloué de la table au-delà de ce que les partitions existantes peuvent prendre en charge.
- Si la capacité d'une partition existante est pleine et qu'un espace de stockage supplémentaire est obligatoire.

La gestion de la partition s'effectue automatiquement à l'arrière-plan et est transparente pour vos applications. Votre table reste disponible tout au long et prend entièrement en charge vos besoins de débit alloué.

Pour en savoir plus, consultez Création de clés de partition.

Des index secondaires globaux dans DynamoDB sont également composés de partitions. Les données d'un index secondaire global sont stockées séparément des données de sa table de base, mais les partitions d'index se comportent de la même manière que les partitions de table.

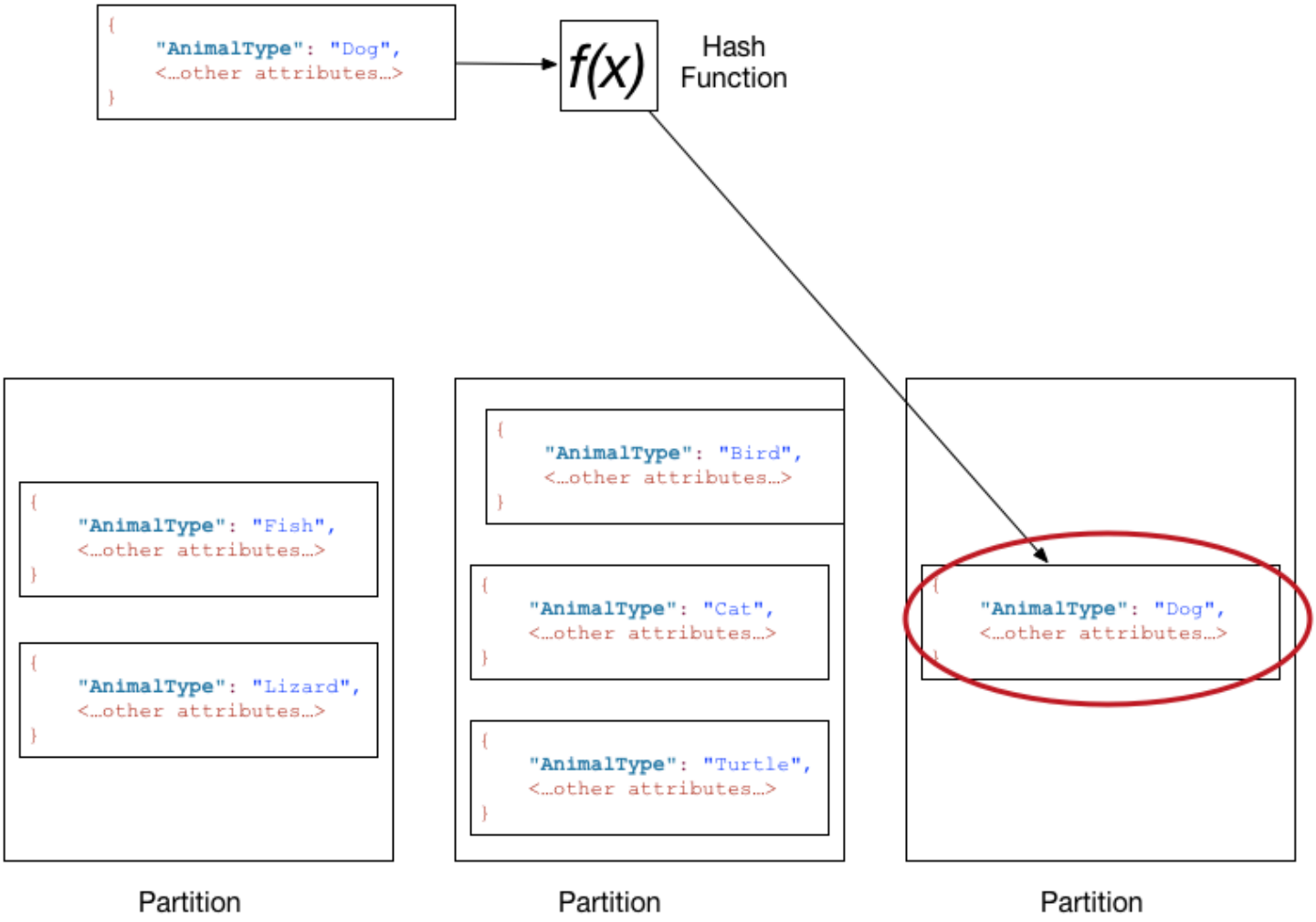
Distribution de données : clé de partition

Si votre table possède une clé primaire simple (clé de partition uniquement), DynamoDB stocke et récupère chaque élément en fonction de sa valeur de clé de partition.

Pour écrire un élément dans la table, DynamoDB utilise la valeur de la clé de partition comme entrée pour une fonction de hachage interne. La valeur de sortie de la fonction de hachage détermine la partition dans laquelle l'élément sera stocké.

Pour lire un élément de la table, vous devez spécifier sa valeur de clé de partition. DynamoDB utilise cette valeur comme entrée pour sa fonction de hachage, donnant la partition dans laquelle se trouve l'élément.

Le schéma suivant montre une table nommée Animaux de compagnie, qui s'étend sur plusieurs partitions. La clé primaire de la table est AnimalType(seul cet attribut clé est affiché). DynamoDB utilise sa fonction de hachage pour déterminer où stocker un nouvel élément, en l'occurrence, en fonction de la valeur de hachage de la chaîne Dog. Notez que les éléments ne sont pas stockés selon un ordre trié. L'emplacement de chaque élément est déterminé par la valeur de hachage de sa clé de partition.



> **Note**
>
> DynamoDB est optimisé pour une distribution uniforme des éléments entre les partitions d'une table, quel que soit le nombre de partitions. Nous vous conseillons de choisir une clé de partition qui peut avoir un grand nombre de valeurs distinctes par rapport au nombre d'éléments de la table.

Distribution de données : clé de partition et clé de tri

Si la table possède une clé primaire composite (clé de partition et clé de tri), DynamoDB calcule la valeur de hachage de la clé de partition de la manière décrite dans Distribution de données : clé de partition. Cependant, il a tendance à conserver les éléments ayant la même valeur de clé de partition proches les uns des autres et triés en fonction de la valeur de l'attribut de la clé de tri. L'ensemble des éléments qui ont la même valeur de clé de partition est nommé collection d'éléments. Les

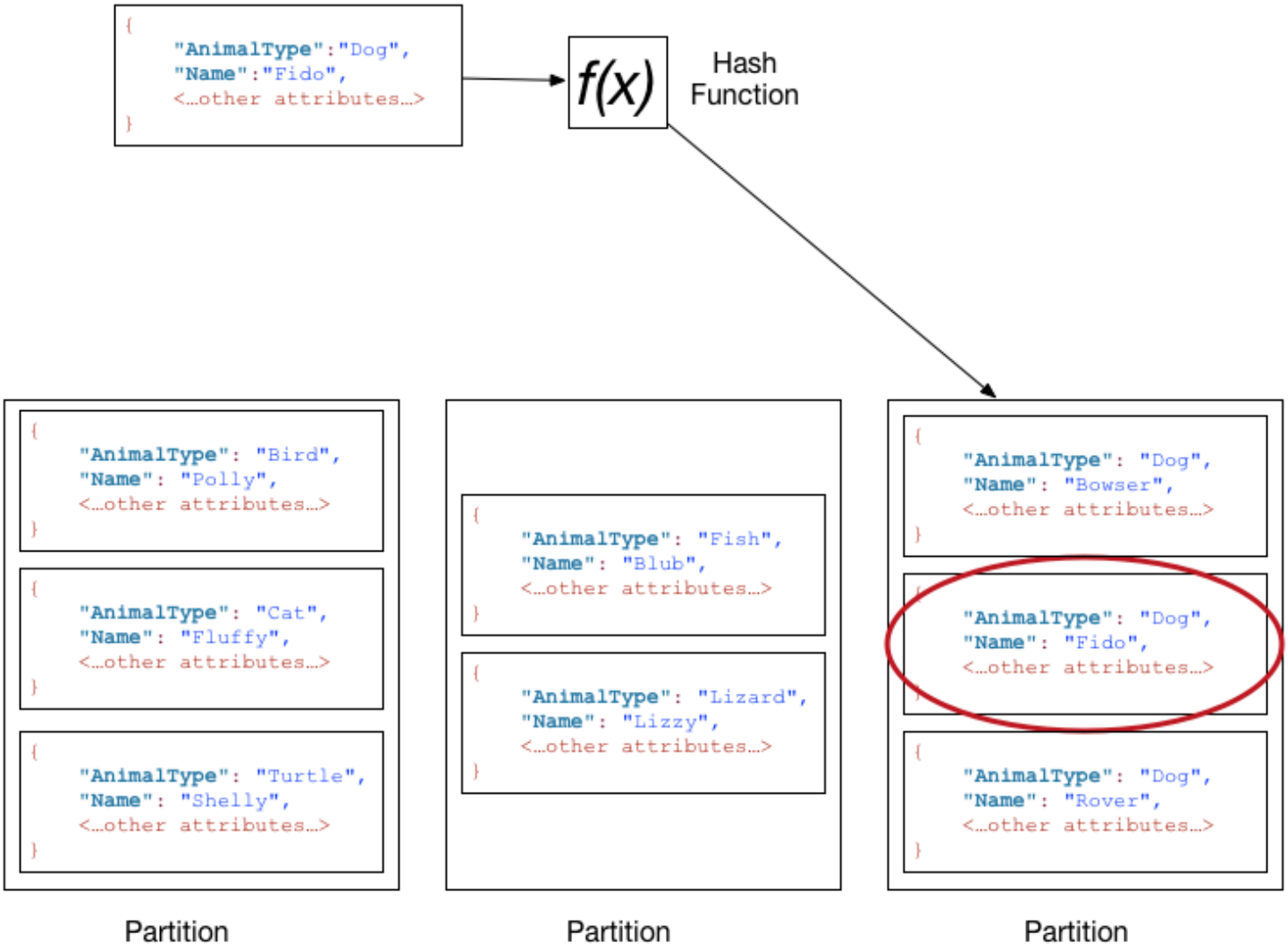
collections d'éléments sont optimisées pour assurer une extraction efficace des gammes d'éléments dans la collection. Si votre table ne possède pas d'index secondaires locaux, DynamoDB répartit automatiquement votre collection d'éléments sur autant de partitions que nécessaire pour stocker les données et optimiser le débit de lecture et d'écriture.

Pour écrire un élément dans la table, DynamoDB calcule la valeur de hachage de la clé de partition pour déterminer quelle partition doit contenir l'élément. Dans cette partition, plusieurs éléments peuvent avoir la même valeur de clé de partition. Ainsi, DynamoDB stocke l'élément parmi les autres éléments possédant la même clé de partition, dans l'ordre croissant de la clé de tri.

Pour lire un élément dans la table, vous devez spécifier les valeurs de ses clés de partition et de tri. DynamoDB calcule la valeur de hachage de la clé de partition, donnant la partition dans laquelle se trouve l'élément.

Vous pouvez lire plusieurs éléments de la table en une seule opération (`Query`) si les éléments souhaités ont la même valeur de clé de partition. DynamoDB renvoie tous les éléments ayant cette valeur de clé de partition. Le cas échéant, vous pouvez appliquer une condition à la clé de tri afin qu'elle retourne uniquement les éléments d'une plage de valeurs donnée.

Supposons que la table Pets possède une clé primaire composite composée de AnimalType(clé de partition) et de nom (clé de tri). Le diagramme suivant montre DynamoDB écrivant un élément avec une valeur de clé de partition Dog et une valeur de clé de tri Fido.



Pour lire cet élément dans la table Pets, DynamoDB calcule la valeur de hachage de Dog, donnant la partition dans laquelle ces éléments sont stockés. DynamoDB analyse ensuite les valeurs d'attribut de clé de tri jusqu'à trouver Fido.

Pour lire tous les éléments marqués AnimalTyped'un Dog, vous pouvez effectuer une `Query` opération sans spécifier de condition de clé de tri. Par défaut, les éléments sont retournés dans l'ordre où ils sont stockés (c'est-à-dire, en ordre croissant par clé de tri). Le cas échéant, vous pouvez demander à la place l'ordre décroissant.

Pour interroger uniquement certains éléments Chien, vous pouvez appliquer une condition à la clé de tri (par exemple, seuls les éléments Chien où Nom commence par une lettre située dans la plage A à K).

> **Note**
>
> Dans une table DynamoDB, il n'existe pas de limite supérieure au nombre de valeurs de clé de tri distinctes par valeur de clé de partition. Si vous avez besoin de stocker plusieurs milliards d'éléments Dog dans la table Pets, DynamoDB alloue automatiquement un stockage suffisant pour gérer cette exigence.

Découvrez comment passer de SQL à NoSQL

Si vous êtes développeur d'applications, il se peut que vous ayez une certaine expérience concernant l'utilisation d'un système de gestion de base de données relationnelle (SGBDR) et d'un langage de recherche structurée (SQL). Lorsque vous commencerez à utiliser le service Amazon DynamoDB, vous constaterez qu'il présente bon nombre de similitudes mais aussi de différences par rapport à SQL. NoSQL est un terme utilisé pour décrire les systèmes de base de données non relationnelle qui sont hautement disponibles, évolutifs et optimisés pour des performances élevées. Au lieu du modèle relationnel, les bases de données NoSQL (comme DynamoDB) utilisent d'autres modèles pour la gestion des données, tels que les paires clé-valeur ou le stockage de documents. Pour plus d'informations, veuillez consulter Présentation de NoSQL.

Amazon DynamoDB prend en charge PartiQL, un langage de requête open source compatible SQL qui vous permet d'interroger les données facilement et de manière efficace, quel que soit leur emplacement ou format de stockage. Avec PartiQL, vous pouvez facilement traiter des données structurées à partir de bases de données relationnelles, des données semi-structurées et imbriquées dans des formats de données ouvertes, et même des données sans schéma dans NoSQL ou des bases de données document qui autorisent différents attributs pour différentes lignes. Pour plus d'informations, consultez Langage de requête PartiQL.

Les sections suivantes décrivent les tâches courantes de base de données, en comparant les instructions SQL et leurs opérations DynamoDB équivalentes.

> **Note**
>
> Les exemples SQL de cette section sont compatibles avec le SGBDR MySQL.
> Les exemples DynamoDB dans cette section affichent le nom de l'opération DynamoDB, ainsi que ses paramètres au format JSON.

Rubriques

- Choix entre système relationnel (SQL) et NoSQL
- Différences entre l'accès à une base de données relationnelle (SQL) et à DynamoDB
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la création d'une table
- Différences entre l'obtention d'informations sur une table à partir d'une base de données relationnelle (SQL) et de DynamoDB
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de l'écriture de données dans une table
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la lecture de données à partir d'une table
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la gestion d'index
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la modification de données dans une table
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la suppression de données dans une table
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la suppression d'une table

Choix entre système relationnel (SQL) et NoSQL

Aujourd'hui, les applications ont des besoins plus exigeants que par le passé. Par exemple, un jeu en ligne peut démarrer avec quelques utilisateurs seulement et une très petite quantité de données. Cependant, si le jeu devient un succès, il peut facilement dépasser les ressources du système de gestion de la base de données sous-jacente. Il est fréquent que les applications web aient des centaines, des milliers ou des millions d'utilisateurs simultanés, avec des téraoctets ou plus de nouvelles données générées par jour. Les bases de données de telles applications doivent gérer des dizaines (voire des centaines) de milliers de lectures et d'écritures par seconde.

Amazon DynamoDB convient parfaitement à ces types de charges de travail. En tant que développeur, vous pouvez commencer à petite échelle et augmenter progressivement votre utilisation au fur et à mesure que votre application gagne en popularité. DynamoDB se met à l'échelle en toute transparence pour gérer un très grand nombre d'utilisateurs et des quantités considérables de données.

Pour en savoir plus sur la modélisation des bases de données relationnelles traditionnelles et sur la façon de l'adapter à DynamoDB, consultez Bonnes pratiques de modélisation des données relationnelles dans DynamoDB.

Le tableau suivant présente quelques différences générales entre un système de gestion de base de données relationnelle (SGBDR) et DynamoDB.

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
Charges de travail optimales	Requêtes ad hoc ; entreposage de données ; traitement analytique en ligne (OLAP).	Applications web, y compris les réseaux sociaux, les jeux, le partage des fichiers multimédia et l'Internet des objets (IoT).
Modèle de données	Le modèle relationnel nécessite un schéma bien défini, dans lequel les données sont normalisées en tables, lignes et colonnes. De plus, toutes les relations sont définies entre tables, colonnes, index et autres éléments de base de données.	DynamoDB est sans schéma. Chaque table doit avoir une clé primaire pour identifier de façon unique chaque élément de données, mais aucune contrainte similaire ne s'applique aux attributs autres que de clé. DynamoDB peut gérer des données structurées ou semi-structurées, y compris des documents JSON.
Accès aux données	SQL constitue la norme pour le stockage et la récupération de données. Les bases de données relationnelles offrent un riche ensemble d'outils pour simplifier le développement d'applications de base de données, mais tous ces outils utilisent SQL.	Vous pouvez utiliser le AWS Management ConsoleAWS CLI, le ou NoSQL WorkBench pour travailler avec DynamoDB et effectuer des tâches ad hoc. PartiQL, un langage de requête compatible SQL, vous permet de sélectionner,

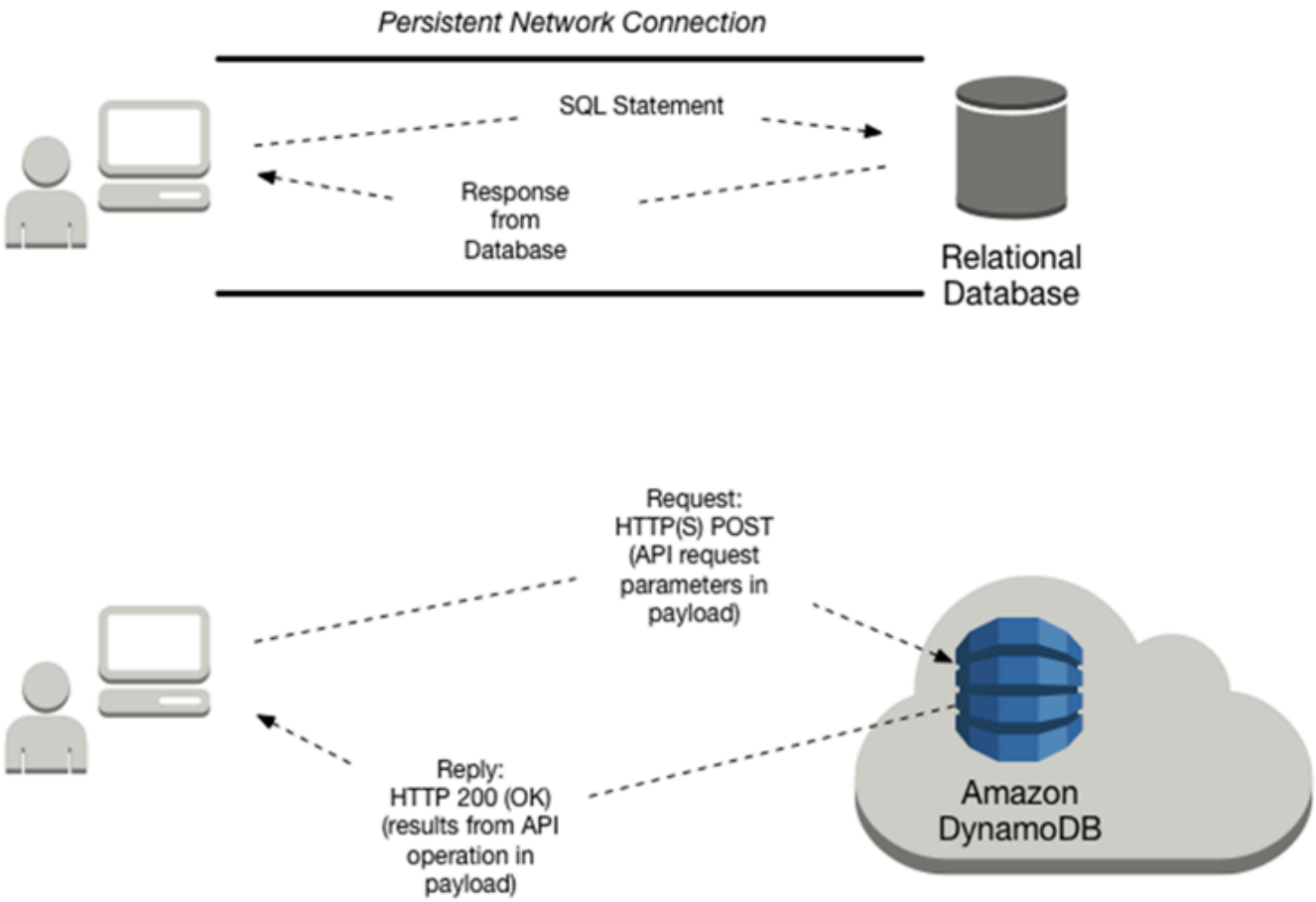
Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
		d’insérer, de mettre à jour et de supprimer des données dans DynamoDB. Les applications peuvent utiliser les kits de développement AWS logiciel (SDKs) pour travailler avec DynamoDB à l'aide d'interfaces basées sur des objets, centrées sur le document ou de bas niveau.
Performances	Les bases de données relationnelles étant optimisées pour le stockage, les performances dépendent généralement du sous-système du disque. Les développeurs et les administrateurs de base de données doivent optimiser les requêtes, les index et les structures de table afin d’atteindre des performances de pointe.	DynamoDB étant optimisé pour le calcul, les performances dépendent principalement du matériel sous-jacent et de la latence réseau. En tant que service géré, DynamoDB vous dispense, ainsi que vos applications, de ces détails d’implémentation, si bien que vous pouvez vous concentrer sur la conception et la génération d’applications robustes hautement performantes.

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
Dimensionnement	Il est plus facile d'agrandir avec un matériel plus rapide. Il est également possible pour les tables de base de données de s'étendre sur plusieurs hôtes d'un système distribué , mais cela nécessite des investissements supplémen taires. Les bases de données relationnelles possèdent une taille maximale pour le nombre de fichiers et leur taille, ce qui impose des limites supérieur es sur l'évolutivité.	DynamoDB est conçu pour monter en charge à l'aide de clusters distribués de matériel. Cette conception permet un débit accru sans augmentat ion de la latence. Les clients spécifient leurs exigences de débit et DynamoDB alloue des ressources suffisantes pour répondre à ces exigences. Il n'y a aucune limite supérieure sur le nombre d'éléments par table, ni sur la taille totale de la table.

Différences entre l'accès à une base de données relationnelle (SQL) et à DynamoDB

Avant que votre application puise accéder à une base de données, elle doit être authentifiée pour garantir que l'application est autorisée à utiliser la base de données. Elle doit être autorisée de manière à ce que l'application ne puisse effectuer que les actions pour lesquelles elle possède des autorisations.

Le schéma suivant illustre l'interaction d'un client avec une base de données relationnelle et avec Amazon DynamoDB.



Le tableau suivant propose plus d'informations sur les tâches liées à l'interaction client.

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
Outils pour accéder à la base de données	La plupart des bases de données relationnelles offrent une interface de ligne de commande (CLI), de telle sorte que vous pouvez saisir des instructions SQL ad hoc et observer les résultats immédiatement.	Dans la plupart des cas, vous écrivez le code de l'application. Vous pouvez également utiliser le Workbench AWS Management Console, the AWS Command Line Interface (AWS CLI) ou NoSQL pour envoyer des requêtes ad hoc à DynamoDB et afficher les résultats. PartiQL, un langage

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
		de requête compatible SQL, vous permet de sélectionner, d'insérer, de mettre à jour et de supprimer des données dans DynamoDB.
Connexion à la base de données	Un programme d'application établit et gère une connexion réseau avec la base de données. Lorsque l'application est terminée, il met fin à la connexion.	DynamoDB est un service web, et les interactions avec celui-ci sont sans état. Les applications n'ont pas besoin de maintenir des connexions réseau permanentes. Au lieu de cela, l'interaction avec DynamoDB se produit via des requêtes et réponses HTTP(S).
Authentification	Une application ne peut pas se connecter à la base de données tant qu'elle n'est pas authentifiée. Le SGBDR peut effectuer l'authentification lui-même ou déléguer cette tâche au système d'exploitation hôte ou à un service d'annuaire.	Chaque demande adressée à DynamoDB doit être accompagnée d'une signature de chiffrement qui authentifie cette demande particulière. Ils AWS SDKs fournissent toute la logique nécessaire pour créer des signatures et des demandes de signature. Pour plus d'informations, consultez la section Signature des demandes d'AWS API dans le Références générales AWS.

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
Autorisation	Les applications ne peuvent exécuter que les actions pour lesquelles elles sont autorisées. Les administrateurs de base de données ou les propriétaires d'applications peuvent utiliser les instructions SQL `GRANT` et `REVOKE` pour contrôler l'accès aux objets de base de données (tels que les tables) et aux données (telles que les lignes d'une table), ou la possibilité d'émettre certaines instructions SQL.	Dans DynamoDB, l'autorisation est gérée Gestion des identités et des accès AWS par (IAM). Vous pouvez écrire une politique IAM pour accorder des autorisations sur une ressource DynamoDB (telle une table), puis autoriser des utilisateurs et des rôles à utiliser cette politique . IAM offre également un contrôle précis des accès pour les éléments de données individuels dans les tables DynamoDB. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour Amazon DynamoDB.
Envoi d'une demande	L'application émet une instruction SQL pour chaque opération de base de données qu'elle souhaite effectuer. À la réception de l'instruction SQL, le SGBDR vérifie sa syntaxe, crée un plan pour effectuer l'opération, puis exécute le plan.	L'application envoie des requêtes HTTP(S) à DynamoDB. Une requête contient le nom de l'opération DynamoDB à effectuer, ainsi que les paramètres de celle-ci. DynamoDB exécute la requête immédiatement.

Caractéristiques	Système de gestion de base de données relationnelle (SGBDR)	Amazon DynamoDB
Réception d'une réponse	Le SGBDR retourne les résultats de l'instruction SQL. S'il y a une erreur, le SGBDR retourne un statut d'erreur et un message.	DynamoDB renvoie une réponse HTTP(S) contenant les résultats de l'opération. En cas d'erreur, DynamoDB renvoie un état et des messages d'erreur HTTP.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la création d'une table

Les tables sont les structures de données fondamentales dans les bases de données relationnelles et dans Amazon DynamoDB. Un système de gestion de base de données relationnelles (SGBDR) vous permet de définir le schéma de la table lorsque vous la créez. En revanche, les tables DynamoDB sont sans schéma. Lorsque vous créez une table, en dehors de la clé primaire, vous n'avez pas besoin de définir d'attributs ou de types de données supplémentaires.

La section suivante compare la méthode de création d'une table avec SQL d'une part et avec DynamoDB d'autre part.

Rubriques

- Création d'une table avec SQL
- Création d'une table avec DynamoDB

Création d'une table avec SQL

Avec SQL, la création d'une table s'effectue via l'instruction `CREATE TABLE`, comme le montre l'exemple suivant.

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
```

```
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

La clé primaire de cette table comprend Artist et SongTitle.

Vous devez définir toutes les colonnes de la table et leurs types de données, ainsi que la clé primaire de la table. (Vous pouvez utiliser l'instruction `ALTER TABLE` pour modifier ces définitions ultérieurement, si nécessaire.)

De nombreuses implémentations SQL vous permettent de définir les spécifications de stockage de votre table, dans le cadre de l'instruction `CREATE TABLE`. Sauf spécification contraire, la table est créée avec les paramètres de stockage par défaut. Dans un environnement de production, un administrateur de base de données peut aider à déterminer les paramètres de stockage optimaux.

Création d'une table avec DynamoDB

Utilisez l'opération `CreateTable` pour créer une table en mode alloué, en spécifiant les paramètres comme suit :

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
```

```
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

La clé primaire de cette table comprend Artist (clé de partition) et SongTitle(clé de tri).

Vous devez fournir les paramètres suivants à `CreateTable` :

- `TableName` – Nom de la table.
- `KeySchema` – Attributs utilisés pour la clé primaire. Pour plus d'informations, consultez Tables, éléments et attributs et Clé primaire.
- `AttributeDefinitions` – Types de données pour les attributs du schéma de clé.
- `ProvisionedThroughput (for provisioned tables)` – Nombre de lectures et d'écritures par seconde nécessaires pour cette table. DynamoDB réserve suffisamment de ressources de stockage et système pour que vos besoins de débit soient toujours satisfaits. Vous pouvez utiliser l'opération `UpdateTable` pour modifier ces éléments ultérieurement, si nécessaire. Vous n'avez pas besoin de spécifier les besoins de stockage d'une table, car DynamoDB gère entièrement l'allocation du stockage.

Différences entre l'obtention d'informations sur une table à partir d'une base de données relationnelle (SQL) et de DynamoDB

Vous pouvez vérifier qu'une table a été créée selon vos spécifications. Dans une base de données relationnelle, la totalité du schéma de la table est affiché. Les tables Amazon DynamoDB étant sans schéma, seuls les attributs de clé primaire sont affichés.

Rubriques

- Obtention d'informations sur une table avec SQL
- Obtention d'informations sur une table dans DynamoDB

Obtention d'informations sur une table avec SQL

La plupart des systèmes de gestion de base de données relationnelle (SGBDR) permettent de décrire la structure d'une table : colonnes, types de données, définition de clé primaire, etc. Il n'existe aucune

méthode standard pour ce faire dans SQL. Cependant, beaucoup de systèmes de base de données offrent une commande `DESCRIBE`. Voici un exemple de fichier de MySQL :

```
DESCRIBE Music;
```

Cette commande renvoie la structure de votre table, avec l'ensemble des noms de colonnes, types de données et tailles.

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

La clé primaire de cette table comprend Artist et SongTitle.

Obtention d'informations sur une table dans DynamoDB

DynamoDB intègre une opération `DescribeTable`, qui est similaire. Le seul paramètre requis est le nom de la table.

```
{
    TableName : "Music"
}
```

La réponse de `DescribeTable` se présente ainsi :

```
{
  "Table": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
```

```
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "Music",
    "KeySchema": [
      {
        "AttributeName": "Artist",
        "KeyType": "HASH"  //Partition key
      },
      {
        "AttributeName": "SongTitle",
        "KeyType": "RANGE"  //Sort key
      }
    ],

    ...
```

`DescribeTable` renvoie également des informations sur les index de la table, les paramètres de débit alloué, le nombre approximatif d'éléments et d'autres métadonnées.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de l'écriture de données dans une table

Les tables des bases de données relationnelles contiennent des lignes de données. Les lignes sont composées de colonnes. Les tables Amazon DynamoDB contiennent des éléments. Les éléments sont composés d'attributs.

Cette section décrit comment écrire une ligne (ou un élément) dans une table.

Rubriques

- Écriture de données dans une table avec SQL
- Écrire de données dans une table dans DynamoDB

Écriture de données dans une table avec SQL

Une table d'une base de données relationnelle est une structure de données à deux dimensions, composée de lignes et de colonnes. Certains systèmes de gestion de base de données fournissent également la prise en charge des données semi-structurées, généralement avec les types de

données natifs JSON ou XML. Cependant, les détails de la mise en œuvre varient selon les fournisseurs.

En SQL, l'ajout d'une ligne à une table s'effectue via l'instruction `INSERT`.

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

La clé primaire de cette table comprend Artist et SongTitle. Vous devez spécifier les valeurs de ces colonnes.

> **Note**
>
> Cet exemple utilise la colonne Tags pour stocker les données semi-structurées sur les chansons de la table Music. La colonne Tags est définie au format TEXTE, ce qui permet de stocker jusqu'à 65 535 caractères dans MySQL.

Écrire de données dans une table dans DynamoDB

Dans Amazon DynamoDB, vous pouvez ajouter un élément à une table à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL).

DynamoDB API

Avec l'API DynamoDB, vous utilisez l'opération `PutItem` pour ajouter un élément à une table.

```
{
    TableName: "Music",
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
```

```
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                  "Smith",
                  "Jones",
                  "Davis"
            ],
            "LengthInSeconds": 214
        }
    }
}
```

La clé primaire de cette table comprend Artist et SongTitle. Vous devez spécifier les valeurs de ces attributs.

Voici quelques éléments clés à savoir sur cet exemple `PutItem` :

- DynamoDB assure une prise en charge native des documents à l'aide de JSON. Cela fait de DynamoDB un outil idéal pour stocker des données semi-structurées telles que des étiquettes. Vous pouvez également récupérer et manipuler les données à partir des documents JSON.
- La table Music ne possède aucun attribut prédéfini, à l'exception de la clé primaire (Artist et SongTitle).
- La plupart des bases de données SQL sont orientées transactions. Lorsque vous émettez une instruction `INSERT`, les modifications des données ne sont pas permanentes jusqu'à ce que vous lanciez une instruction `COMMIT`. Avec Amazon DynamoDB, les effets d'une opération `PutItem` sont permanents lorsque DynamoDB répond par un code d'état HTTP 200 (OK).

Voici quelques autres exemples de `PutItem`.

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
```

```
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
```

```
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

> **Note**
>
> En plus de `PutItem`, DynamoDB prend en charge une opération `BatchWriteItem` pour écrire plusieurs éléments à la fois.

PartiQL for DynamoDB

Avec PartiQL, vous utilisez l'opération `ExecuteStatement` pour ajouter un élément à une table, à l'aide de l'instruction PartiQL `Insert`.

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

La clé primaire de cette table comprend Artist et SongTitle. Vous devez spécifier les valeurs de ces attributs.

> **Note**
>
> Pour des exemples de code utilisant `Insert` et `ExecuteStatement`, consultez Instructions d'insertion de PartiQL pour DynamoDB.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la lecture de données à partir d'une table

Avec SQL, vous utilisez l'instruction `SELECT` pour extraire une ou plusieurs lignes d'une table. Vous utilisez la clause `WHERE` pour déterminer les données qui vous sont renvoyées.

Cela est différent d'Amazon DynamoDB, qui propose les opérations suivantes pour la lecture de données :

- `ExecuteStatement` récupère un ou plusieurs éléments d'une table. `BatchExecuteStatement` récupère plusieurs éléments de tables différentes en une seule opération. Ces deux opérations utilisent PartiQL, un langage de requête compatible SQL.
- `GetItem` – Extrait un élément unique d'une table. C'est le moyen le plus efficace de lire un élément unique, car il fournit un accès direct à l'emplacement physique de l'élément. (DynamoDB fournit également l'opération `BatchGetItem` permettant d'effectuer jusqu'à 100 appels `GetItem` en une seule opération.)
- `Query` – Extrait tous les éléments ayant une clé de partition spécifique. Au sein de ces éléments, vous pouvez appliquer une condition à la clé de tri et récupérer uniquement un sous-ensemble des données. `Query` fournit un accès rapide et efficace aux partitions où les données sont stockées. (Pour plus d'informations, consultez Partitions et distribution des données dans DynamoDB.)
- `Scan` – Extrait tous les éléments de la table spécifiée. (Cette opération ne doit pas être utilisée avec les grandes tables, car elle peut consommer d'importantes quantités de ressources système.)

> **Note**
>
> Avec une base de données relationnelle, vous pouvez utiliser l'instruction `SELECT` pour joindre les données de plusieurs tables et retourner les résultats. Les jointures sont fondamentales pour le modèle relationnel. Pour garantir l'efficacité des jointures, les performances de la base de données et de ses applications doivent être ajustées en permanence. DynamoDB est une base de données NoSQL non relationnelle qui ne prend pas en charge les jointures de tables. Au lieu de cela, les applications lisent les données d'une seule table à la fois.

Les sections suivantes décrivent les différents cas d'utilisation de lecture des données et comment exécuter ces tâches avec une base de données relationnelle et avec DynamoDB.

Rubriques

- Différences dans la lecture d'un élément à l'aide de sa clé primaire
- Différences dans l'interrogation d'une table
- Différences dans l'analyse d'une table

Différences dans la lecture d'un élément à l'aide de sa clé primaire

Un modèle d'accès commun aux bases de données consiste à lire un seul élément d'une table. Vous devez spécifier la clé primaire de l'élément que vous voulez.

Rubriques

- Lecture d'un élément à partir de sa clé primaire avec SQL
- Lecture d'un élément à partir de sa clé primaire dans DynamoDB

Lecture d'un élément à partir de sa clé primaire avec SQL

En SQL, l'extraction de données d'une table s'effectue à l'aide de l'instruction `SELECT`. Vous pouvez demander une ou plusieurs colonnes du résultat (ou la totalité, si vous utilisez l'opération `*`). La clause `WHERE` détermine les lignes à retourner.

L'instruction `SELECT` suivante permet d'extraire une seule ligne de la table Music. La clause `WHERE` spécifie les valeurs de clé primaire.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Vous pouvez modifier cette requête pour récupérer uniquement un sous-ensemble des colonnes.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Notez que la clé primaire de ce tableau est composée de Artist et SongTitle.

Lecture d'un élément à partir de sa clé primaire dans DynamoDB

Dans Amazon DynamoDB, vous pouvez lire un élément à partir d'une table à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL).

DynamoDB API

Avec l'API DynamoDB, vous utilisez l'opération `PutItem` pour ajouter un élément à une table.

DynamoDB propose l'opération `GetItem` pour extraire un élément à partir de sa clé primaire. L'opération `GetItem` est très efficace, car elle donne directement accès à l'emplacement physique de l'élément. (Pour plus d'informations, consultez Partitions et distribution des données dans DynamoDB.)

Par défaut, `GetItem` retourne l'élément entier avec tous ses attributs.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

Vous pouvez ajouter un paramètre `ProjectionExpression` pour renvoyer uniquement certains des attributs.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Notez que la clé primaire de ce tableau est composée de Artist et SongTitle.

L'opération `GetItem` de DynamoDB est très efficace. Elle utilise les valeurs de clé primaire pour déterminer l'emplacement exact de stockage de l'élément en question et le récupère directement de cet emplacement. L'instruction SQL `SELECT` est efficace de la même façon, en ce qui concerne la récupération d'éléments par valeurs de clé primaire.

L'instruction SQL `SELECT` prend en charge de nombreux types de requêtes et d'analyses de table. DynamoDB propose une fonctionnalité similaire avec ses opérations `Query` et `Scan`, qui sont décrites dans Différences dans l'interrogation d'une table et Différences dans l'analyse d'une table.

L'instruction SQL `SELECT` permet d'effectuer des jointures de tables, ce qui vous permet de récupérer les données de plusieurs tables à la fois. Les jointures sont plus efficaces lorsque les

tables de base de données sont normalisées et que les relations entre les tables sont claires. Toutefois, si vous joignez un trop grand nombre de tables dans une instruction `SELECT`, les performances de l'application peuvent en être affectées. Vous pouvez contourner ces problèmes en utilisant la réplication de base de données, les vues matérialisées ou les réécritures de requête.

DynamoDB est une base de données non relationnelle ne prenant pas en charge les jointures de tables. Si vous migrez une application existante à partir d'une base de données relationnelle vers DynamoDB, vous devez dénormaliser votre modèle de données pour éliminer la nécessité de jointures.

PartiQL for DynamoDB

Avec PartiQL, vous utilisez l'opération `ExecuteStatement` pour lire un élément d'une table, à l'aide de l'instruction PartiQL `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Notez que la clé primaire de ce tableau est composée de Artist et SongTitle.

> **Note**
>
> L'instruction PartiQL select permet également d'interroger ou d'analyser une table DynamoDB

Pour des exemples de code utilisant `Select` et `ExecuteStatement`, consultez Instructions PartiQL de sélection pour DynamoDB.

Différences dans l'interrogation d'une table

Un autre modèle commun d'accès est la lecture de plusieurs éléments d'une table, selon les critères de votre requête.

Rubriques

- Interrogation d'une table avec SQL
- Interrogation d'une table dans DynamoDB

Interrogation d'une table avec SQL

Lorsque vous utilisez SQL, l'instruction `SELECT` vous permet d'interroger les colonnes clé, les colonnes non-clé ou toute autre combinaison. La clause `WHERE` détermine quelles lignes sont retournées, comme illustré dans les exemples suivants.

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

Notez que la clé primaire de ce tableau est composée de Artist et SongTitle.

Interrogation d'une table dans DynamoDB

Dans Amazon DynamoDB, vous pouvez interroger un élément à partir d'une table à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL).

DynamoDB API

> Avec Amazon DynamoDB, vous pouvez utiliser l'opération `Query` pour extraire des données de façon similaire. L'opération `Query` fournit un accès rapide et efficace aux emplacements

physiques où sont stockées les données. Pour de plus amples informations, veuillez consulter Partitions et distribution des données dans DynamoDB.

Vous pouvez utiliser `Query` avec n'importe quel(le) table ou index secondaire. Vous devez spécifier une condition d'égalité pour la valeur de la clé de partition et, le cas échéant, fournir une autre condition pour l'attribut de la clé de tri s'il est défini.

Le paramètre `KeyConditionExpression` spécifie les valeurs clés que vous souhaitez interroger. Vous pouvez utiliser un paramètre `FilterExpression` facultatif pour supprimer des résultats certains éléments avant qu'ils ne vous soient retournés.

Dans DynamoDB, vous devez utiliser des `ExpressionAttributeValues` comme espaces réservés dans des paramètres d'expression (tels que `KeyConditionExpression` et `FilterExpression`). C'est analogue à l'utilisation des variables de liaison des bases de données relationnelles, où vous remplacez les valeurs réelles dans l'instruction `SELECT` au moment de l'exécution.

Notez que la clé primaire de ce tableau est composée de Artist et SongTitle.

Voici quelques exemples d'actions `Query` de DynamoDB.

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

PartiQL for DynamoDB

Avec PartiQL, vous pouvez exécuter une requête en utilisant l'opération `ExecuteStatement` et l'instruction `Select` sur la clé de partition.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

Une telle utilisation de l'instruction `SELECT` renvoie tous les morceaux associés à cet `Artist` spécifique.

Pour des exemples de code utilisant `Select` et `ExecuteStatement`, consultez Instructions PartiQL de sélection pour DynamoDB.

Différences dans l'analyse d'une table

Dans SQL, une instruction `SELECT` sans une clause `WHERE` retourne chaque ligne d'une table. Dans Amazon DynamoDB, l'opération `Scan` effectue la même chose. Dans les deux cas, vous pouvez extraire tous les éléments, ou seulement certains d'entre eux.

Que vous utilisiez une base de données SQL ou NoSQL, les analyses doivent être utilisées avec modération, car elles peuvent consommer d'importantes quantités de ressources système. Parfois, une analyse est appropriée (analyse d'une petite table, par exemple) ou inévitable (exportation en bloc de données, par exemple). Cependant, en règle générale, vous devez concevoir vos applications de façon à éviter d'exécuter des analyses. Pour de plus amples informations, veuillez consulter Interrogation de tables dans DynamoDB.

> **Note**
>
> L'exportation en masse crée également au moins un fichier par partition. Tous les éléments de chaque fichier proviennent de l'espace de clés haché de cette partition particulière.

Rubriques

- Analyse d'une table avec SQL
- Analyse d'une table dans DynamoDB

Analyse d'une table avec SQL

Lorsque vous utilisez SQL, vous pouvez analyser une table et en extraire toutes les données en utilisant une instruction `SELECT` sans spécifier de clause `WHERE`. Vous pouvez demander une ou plusieurs colonnes dans le résultat. Ou, vous pouvez demander leur totalité si vous utilisez le caractère générique (*).

Voici des exemples d'utilisation d'une déclaration `SELECT`.

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

Analyse d'une table dans DynamoDB

Dans Amazon DynamoDB, vous pouvez analyser une table à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL).

DynamoDB API

Grâce à l'API DynamoDB, vous utilisez l'opération `Scan` pour renvoyer un ou plusieurs éléments et attributs d'élément en accédant à chaque élément dans une table ou un index secondaire.

```
// Return all of the data in the table
{
    TableName:  "Music"
```

```
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

L'opération `Scan` propose également un paramètre `FilterExpression` qui permet d'écarter les éléments qui ne doivent pas figurer dans les résultats. Une `FilterExpression` est appliquée une fois l'analyse terminée, avant que les résultats vous soient renvoyés. Cela n'est pas recommandé avec des tables volumineuses, car vous continuez d'être facturé pour l'opération `Scan` complète, même si seuls quelques éléments correspondants sont renvoyés.

PartiQL for DynamoDB

Avec PartiQL, vous procédez à une analyse à l'aide de l'opération `ExecuteStatement`, qui permet de renvoyer l'ensemble du contenu d'une table utilisant l'instruction `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Notez que cette instruction renvoie tous les éléments de la table Music.

Pour des exemples de code utilisant `Select` et `ExecuteStatement`, consultez Instructions PartiQL de sélection pour DynamoDB.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la gestion d'index

Les index donnent accès à d'autres modèles de requête et peuvent accélérer les requêtes. Cette section compare la création et l'utilisation des index dans SQL et Amazon DynamoDB.

Que vous utilisiez une base de données relationnelle ou DynamoDB, vous devez recourir à la création d'index judicieusement. Chaque fois qu'une écriture se produit sur une table, tous les index de la table doivent être mis à jour. Dans un environnement à forte densité d'écritures avec de grandes tables, cette opération peut consommer d'importantes quantité de ressources système. Dans un environnement en lecture seule ou principalement en lecture, ce risque est moins important.

Cependant, vous devez vous assurer que les index sont effectivement utilisés par votre application et pas simplement qu'ils occupent de l'espace.

Rubriques

- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la création d'un index
- Différences entre une base de données relationnelle (SQL) et DynamoDB lors de l'interrogation et de l'analyse d'un index

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la création d'un index

Comparez l'instruction `CREATE INDEX` dans SQL avec l'opération `UpdateTable` dans Amazon DynamoDB.

Rubriques

- Création d'un index avec SQL
- Création d'un index dans DynamoDB

Création d'un index avec SQL

Dans une base de données relationnelle, un index est une structure de données qui vous permet d'exécuter des requêtes rapides sur différentes colonnes d'une table. Vous pouvez utiliser l'instruction SQL `CREATE INDEX` pour ajouter un index à une table existante, en spécifiant les colonnes à indexer. Une fois que l'index a été créé, vous pouvez interroger les données de la table comme à l'habitude, mais maintenant la base de données peut utiliser l'index pour retrouver rapidement les lignes spécifiées dans le tableau au lieu d'analyser la totalité de la table.

Une fois que vous créez un index, la base de données le gère automatiquement. Chaque fois que vous modifiez les données de la table, l'index est automatiquement modifié pour refléter les modifications de la table.

Dans MySQL, il vous faudrait créer un index semblable au suivant.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Création d'un index dans DynamoDB

Dans DynamoDB, vous pouvez créer et utiliser un index secondaire à des fins similaires.

Les index dans DynamoDB sont différents de leurs équivalents relationnels. Lorsque vous créez un index secondaire, vous devez spécifier ses attributs de clé, c'est-à-dire une clé de partition et une clé de tri. Après avoir créé l'index secondaire, vous pouvez effectuer une opération `Query` ou `Scan` sur celui-ci, comme vous le feriez avec une table. DynamoDB n'ayant pas d'optimiseur de requête, un index secondaire n'est utilisé que lorsque vous effectuez une opération `Query` ou `Scan` sur celui-ci.

DynamoDB prend en charge deux types d'index :

- Index secondaires globaux – La clé primaire de l'index peut être constituée de deux attributs quelconques de sa table.
- Index secondaires locaux – La clé de partition de l'index doit être la même que la clé de partition de sa table. Cependant, la clé de tri peut être n'importe quel autre attribut.

DynamoDB veille à ce que les données dans un index secondaire soient éventuellement cohérentes avec sa table. Vous pouvez demander des opérations `Query` ou `Scan` fortement cohérentes sur une table ou un index secondaire local. Cependant, les index secondaires globaux ne prennent en charge que la cohérence éventuelle.

Vous pouvez ajouter un index secondaire global à une table existante à l'aide de l'opération `UpdateTable` et en spécifiant `GlobalSecondaryIndexUpdates`.

```
{
    TableName: "Music",
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
                Projection: {
                    "ProjectionType": "ALL"
```

```
                },
                ProvisionedThroughput: {                                    // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

Vous devez fournir les paramètres suivants à `UpdateTable` :

- `TableName` – Table à laquelle l'index sera associé.
- `AttributeDefinitions` – Types de données pour les attributs de schéma de clé de l'index.
- `GlobalSecondaryIndexUpdates` – Détails sur l'index que vous souhaitez créer :
  - `IndexName` – Nom pour l'index.
  - `KeySchema` – Attributs utilisés pour la clé primaire de l'index.
  - `Projection` – Attributs de la table qui sont copiés dans l'index. Dans ce cas, `ALL` signifie que tous les attributs sont copiés.
  - `ProvisionedThroughput (for provisioned tables)` – Nombre de lectures et d'écritures par seconde nécessaires pour ce index. (Cette information est distincte des paramètres de débit alloué de la table.)

Une partie de cette opération implique le renvoi de données de la table vers le nouvel index. Pendant le renvoi, la table demeure disponible. Cependant, l'index n'est pas prêt jusqu'à ce que son attribut `Backfilling` passe de true à false. Vous pouvez utiliser l'opération `DescribeTable` pour afficher cet attribut.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de l'interrogation et de l'analyse d'un index

Comparez l'interrogation et l'analyse d'un index à l'aide de l'instruction SELECT dans SQL avec les opérations `Query` et `Scan` dans Amazon DynamoDB.

Rubriques

- Interrogation et analyse d'un index avec SQL
- Interrogation et analyse d'un index dans DynamoDB

Interrogation et analyse d'un index avec SQL

Dans une base de données relationnelle, vous n'utilisez pas directement les index. Au lieu de cela, vous interrogez les tables à l'aide d'instructions `SELECT` et l'optimiseur de requête peut utiliser tous les index.

Un optimiseur de requête est un composant d'un système de gestion de base de données relationnelle (SGBDR) qui évalue les index disponibles et détermine s'ils peuvent être utilisés pour accélérer une requête. Si tel est le cas, le SGBDR accède d'abord à l'index, puis l'utilise pour rechercher les données dans la table.

Voici quelques instructions SQL qui peuvent être utilisées GenreAndPriceIndexpour améliorer les performances. Nous supposons que la table Music possède suffisamment de données pour que l'optimiseur de requête décide d'utiliser cet index, plutôt que de simplement analyser la totalité de la table.

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
WHERE Genre = 'Country' AND Price < 0.50;
```

Interrogation et analyse d'un index dans DynamoDB

Dans DynamoDB, vous effectuez les opérations `Query` et `Scan` directement sur l'index, de la même manière que vous le feriez sur une table. Vous pouvez interroger ou analyser l'index à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL). Vous devez spécifier à la fois `TableName` et `IndexName`.

Voici quelques requêtes effectuées GenreAndPriceIndexdans DynamoDB. (Le schéma de clé de l'index est composé de Genre et de Price.)

DynamoDB API

```
// All of the rock songs

{
```

```
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

Cet exemple utilise une `ProjectionExpression` pour indiquer que vous souhaitez que certains attributs seulement, et non leur totalité, apparaissent dans les résultats.

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

Ce qui suit est un scan en cours GenreAndPriceIndex.

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Avec PartiQL, vous utilisez l'instruction PartiQL `Select` pour interroger et analyser l'index.

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

Ce qui suit est un scan en cours GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Pour obtenir des exemples de code utilisant `Select`, consultez Instructions PartiQL de sélection pour DynamoDB.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la modification de données dans une table

Le langage SQL fournit l'instruction `UPDATE` pour modifier les données. Amazon DynamoDB utilise l'opération `UpdateItem` pour accomplir des tâches similaires.

Rubriques

- Modification de données dans une table avec SQL
- Modification de données dans une table dans DynamoDB

Modification de données dans une table avec SQL

En SQL, la modification d'une ou plusieurs lignes s'effectue via l'instruction `UPDATE`. La clause `SET` spécifie de nouvelles valeurs pour une ou plusieurs colonnes, et la clause `WHERE` détermine les lignes qui sont modifiées. Voici un exemple.

```
UPDATE Music
SET RecordLabel = 'Global Records'
```

```
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Si aucune ligne ne correspond à la clause `WHERE`, l'instruction `UPDATE` n'a aucun effet.

Modification de données dans une table dans DynamoDB

Dans DynamoDB, vous pouvez modifier un élément unique à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL). Si vous voulez modifier plusieurs éléments, vous devez utiliser plusieurs opérations.

DynamoDB API

Avec l'API DynamoDB, vous devez utiliser l'opération `UpdateItem` pour modifier un élément unique.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

Vous devez spécifier les attributs `Key` de l'élément à modifier ainsi qu'une `UpdateExpression` pour spécifier les valeurs des attributs. `UpdateItem` se comporte comme une opération « upsert » (mise à jour/insertion). L'élément est mis à jour s'il existe dans la table, mais dans le cas contraire, un nouvel élément est ajouté (inséré).

`UpdateItem` prend en charge les écritures conditionnelles, où l'opération aboutit uniquement si un attribut `ConditionExpression` a la valeur true. Par exemple, l'opération `UpdateItem` suivante n'effectue pas la mise à jour, sauf si le prix de la chanson est supérieur ou égal à 2,00.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
```

```
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

`UpdateItem` prend également en charge les compteurs atomiques, ou les attributs de type `Number` qui peuvent être incrémentés ou décrémentés. Les compteurs atomiques sont similaires en de nombreuses façons aux générateurs de séquence, aux colonnes d'identité ou aux champs d'incrémentation automatique dans les bases de données SQL.

Voici un exemple d'opération `UpdateItem` pour initialiser un nouvel attribut (Plays) et suivre le nombre de fois qu'une chanson a été écoutée.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

Le paramètre `ReturnValues` est défini sur `UPDATED_NEW`, qui retourne les nouvelles valeurs de tous les attributs qui ont été mis à jour. Dans ce cas, la valeur 0 (zéro) est retournée.

Chaque fois que quelqu'un écoute cette chanson, nous pouvons utiliser l'opération `UpdateItem` suivante pour incrémenter la valeur Plays d'une unité.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
```

```
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

PartiQL for DynamoDB

Avec PartiQL, vous utilisez l'opération `ExecuteStatement` pour modifier un élément d'une table, à l'aide de l'instruction PartiQL `Update`.

La clé primaire de cette table comprend Artist et SongTitle. Vous devez spécifier les valeurs de ces attributs.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Vous pouvez également modifier plusieurs champs à la fois, tel qu'illustré dans l'exemple suivant.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` prend également en charge les compteurs atomiques, ou les attributs de type `Number` qui peuvent être incrémentés ou décrémentés. Les compteurs atomiques sont similaires en de nombreuses façons aux générateurs de séquence, aux colonnes d'identité ou aux champs d'incrémentation automatique dans les bases de données SQL.

Voici un exemple d'instruction `Update` destinée à initialiser un nouvel attribut (Plays (Lectures)) et à suivre le nombre de fois où un morceau a été joué.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Chaque fois que quelqu'un écoute ce morceau, nous pouvons utiliser l'instruction `Update` suivante pour incrémenter la valeur Plays (Lectures) d'une unité.

```
UPDATE Music
```

```
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Pour des exemples de code utilisant `Update` et `ExecuteStatement`, consultez Instructions de mise à jour de PartiQL pour DynamoDB.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la suppression de données dans une table

Dans SQL, l'instruction `DELETE` supprime une ou plusieurs lignes d'une table. Amazon DynamoDB utilise l'opération `DeleteItem` pour supprimer un élément à la fois.

Rubriques

- Suppression de données d'une table avec SQL
- Suppression de données d'une table dans DynamoDB

Suppression de données d'une table avec SQL

Dans SQL, vous utilisez l'instruction `DELETE` pour supprimer une ou plusieurs lignes. La clause `WHERE` détermine les lignes que vous voulez modifier. Voici un exemple.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Vous pouvez modifier la clause `WHERE` pour supprimer plusieurs lignes. Par exemple, vous pouvez supprimer tous les morceaux d'un artiste particulier, comme illustré dans l'exemple ci-après :

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Suppression de données d'une table dans DynamoDB

Dans DynamoDB, vous pouvez supprimer un élément unique à l'aide de l'API DynamoDB ou de PartiQL (langage de requête compatible SQL). Si vous voulez modifier plusieurs éléments, vous devez utiliser plusieurs opérations.

DynamoDB API

Avec l'API DynamoDB, vous devez utiliser l'opération `DeleteItem` pour supprimer les données d'une table, un élément à la fois. Vous devez spécifier les valeurs de clé primaire de l'élément.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Outre `DeleteItem`, Amazon DynamoDB prend en charge une opération `BatchWriteItem` pour supprimer plusieurs éléments à la fois.

`DeleteItem` prend en charge les écritures conditionnelles, où l'opération aboutit uniquement si un attribut `ConditionExpression` a la valeur true. Par exemple, l'`DeleteItem`opération suivante supprime l'élément uniquement s'il possède un RecordLabelattribut.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Avec PartiQL, vous utilisez l'instruction `Delete` via l'opération `ExecuteStatement` afin de supprimer des données d'une table, un élément à la fois. Vous devez spécifier les valeurs de clé primaire de l'élément.

La clé primaire de cette table comprend Artist et SongTitle. Vous devez spécifier les valeurs de ces attributs.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Vous pouvez également spécifier des conditions supplémentaires pour l'opération. L'opération `DELETE` suivante ne supprime l'élément que s'il a plus de 11 Awards (Récompenses).

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Pour des exemples de code utilisant `DELETE` et `ExecuteStatement`, consultez Instructions de suppression de PartiQL pour DynamoDB.

Différences entre une base de données relationnelle (SQL) et DynamoDB lors de la suppression d'une table

Dans SQL, vous utilisez l'instruction `DROP TABLE` pour supprimer une table. Dans Amazon DynamoDB, vous utilisez l'opération `DeleteTable`.

Rubriques

- Suppression d'une table avec SQL
- Suppression d'une table dans DynamoDB

Suppression d'une table avec SQL

Lorsque vous n'avez plus besoin d'une table et que vous voulez la supprimer définitivement, vous devez utiliser l'instruction `DROP TABLE` en SQL.

```
DROP TABLE Music;
```

Une fois qu'une table est supprimée, elle ne peut pas être récupérée. (Certaines bases de données relationnelles vous permettent réellement d'annuler une opération `DROP TABLE`, mais il s'agit d'une fonctionnalité propre au fournisseur et qui n'est pas largement appliquée.)

Suppression d'une table dans DynamoDB

Dans DynamoDB, `DeleteTable` est une opération similaire. Dans l'exemple suivant, la table est supprimée de façon définitive.

```
{
    TableName: "Music"
}
```

Ressources et outils de formation Amazon DynamoDB

Vous pouvez utiliser les ressources supplémentaires suivantes pour comprendre et utiliser DynamoDB.

Rubriques

- Outils de codage et de visualisation
- Articles de conseils prescriptifs
- Articles du Centre de connaissances
- Articles de blog, référentiels et guides
- Présentations de la modélisation des données et des modèles de conception
- Formation

Outils de codage et de visualisation

Vous pouvez utiliser les outils de codage et de visualisation suivants avec DynamoDB :

- NoSQL Workbench pour Amazon DynamoDB : outil visuel unifié qui vous aide à concevoir, créer, interroger et gérer des tables DynamoDB. Il fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement de requêtes.
- Dynobase : outil de bureau qui facilite la visualisation et l'utilisation de vos tables DynamoDB, la création de code d'application et la modification de registres avec une validation en temps réel.
- DynamoDB Toolbox — Un projet de Jeremy Daly qui fournit des utilitaires utiles pour travailler avec la modélisation des données et Node.js. JavaScript
- DynamoDB Streams Processor : outil simple facilitant l'utilisation de DynamoDB streams.

Articles de conseils prescriptifs

AWS Prescriptive Guidance fournit des stratégies, des guides et des modèles éprouvés pour vous aider à accélérer vos projets. Ces ressources ont été développées par des experts en AWS technologie et la communauté mondiale de AWS partenaires, sur la base de leurs années d'expérience à aider les clients à atteindre leurs objectifs commerciaux.

Modélisation et migration des données

- Modèle de données hiérarchiques dans DynamoDB
- Modélisation de données avec DynamoDB
- Migrer une base de données Oracle vers DynamoDB à l'aide de AWS DMS

Tableaux globaux

- Utilisation des tables globales Amazon DynamoDB

Serverless (Sans serveur)

- Implémentez le modèle de saga sans serveur avec AWS Step Functions

Architecture SaaS

- Gestion des locataires sur plusieurs produits SaaS sur un seul plan de contrôle
- Intégration des locataires dans l’architecture SaaS pour le modèle de silo avec C# et AWS CDK

Protection et transfert des données

- Configuration de l’accès intercompte à Amazon DynamoDB
- Options de copie complète des tables pour DynamoDB
- Stratégie de reprise après sinistre pour les bases de données sur AWS

Miscellaneous

- Aide pour appliquer le balisage dans DynamoDB

Démonstrations vidéo de conseils prescriptifs

- Utilisation d'une architecture sans serveur pour créer des pipelines de données
- Novartis – Buying Engine : portail d'approvisionnement basé sur l'IA
- Veritiv : utilisez Insights pour prévoir la demande de vente sur AWS les lacs de données
- mimik : le cloud Edge hybride s'appuie sur le support du AWS maillage des microservices Edge
- Récupération de données de modification avec Amazon DynamoDB

Pour d'autres articles et vidéos sur les conseils prescriptifs pour DynamoDB, consultez Conseils prescriptifs.

Articles du Centre de connaissances

Les articles et vidéos du AWS Knowledge Center couvrent les questions et demandes les plus fréquentes que nous recevons de la part des AWS clients. Vous trouverez ci-dessous quelques articles récents du Centre de connaissances sur des tâches spécifiques liées à DynamoDB :

Optimisation des coûts

- Comment optimiser les coûts avec Amazon DynamoDB ?

Limitation et latence

- Comment puis-je résoudre les problèmes de latence élevée sur une table Amazon DynamoDB ?
- Pourquoi ma table DynamoDB est-elle limitée ?
- Pourquoi ma table DynamoDB à la demande est-elle limitée ?

Pagination

- Comment implémenter la pagination dans DynamoDB ?

Transactions

- Pourquoi mon appel d'API `TransactWriteItems` échoue-t-il dans DynamoDB ?

Résolution des problèmes

- Comment résoudre les problèmes liés à l'autoscaling de DynamoDB ?
- Comment résoudre les erreurs HTTP 4XX dans DynamoDB ?

Pour des articles et des vidéos supplémentaires sur DynamoDB, consultez les articles du Centre de connaissances.

Articles de blog, référentiels et guides

Outre le Guide du développeur DynamoDB, il existe de nombreuses ressources utiles pour utiliser DynamoDB. Voici une sélection d'articles de blog, de référentiels et de guides relatifs à l'utilisation de DynamoDB :

- AWSréférentiel d'exemples de code DynamoDB dans AWS différents langages du SDK : Node.js, Java, Python, .Net , Go et Rust.
- Le livre DynamoDB : guide complet d' DeBrieAlex qui enseigne une approche stratégique de la modélisation des données avec DynamoDB.
- Guide DynamoDB : guide ouvert d' DeBrieAlex qui présente les concepts de base et les fonctionnalités avancées de la base de données DynamoDB NoSQL.
- Comment passer de RDBMS à DynamoDB en 20 étapes faciles : liste d'étapes utiles pour l'apprentissage de la modélisation de données par Jeremy Daly.
- Aide-mémoire JavaScript DocumentClient DynamoDB : aide-mémoire destiné à vous aider à créer des applications avec DynamoDB dans un fichier Node.js ou un environnement. JavaScript
- Vidéos sur le concept de base de DynamoDB : cette playlist couvre de nombreux concepts de base de DynamoDB.

Présentations de la modélisation des données et des modèles de conception

Vous pouvez utiliser les ressources suivantes sur la modélisation des données et les modèles de conception pour tirer le meilleur parti de DynamoDB :

- AWS re:Invent 2019 : Modélisation des données avec DynamoDB
  - Une conférence d'Alex DeBrie qui vous aide à vous familiariser avec les principes de la modélisation des données DynamoDB.
- AWS re:Invent 2020 : Modélisation des données avec DynamoDB — Partie 1

- AWS re:Invent 2020 : Modélisation des données avec DynamoDB — Partie 2
- AWS re:Invent 2017 : Modèles de design avancés
- AWS re:Invent 2018 : modèles de design avancés
- AWS re:Invent 2019 : modèles de conception avancés
  - Jeremy Daly partage ses 12 points clés à retenir au cours de cette session.
- AWS re:Invent 2020 : modèles de conception avancée DynamoDB – Partie 1
- AWS re:Invent 2020 : Modèles de conception avancés de DynamoDB — Partie 2
- Heures de bureau DynamoDB sur Twitch

> **Note**
>
> Chaque session couvre différents exemples et cas d'utilisation.

Formation

Il existe de nombreux cours de formation et options pédagogiques pour en savoir plus sur DynamoDB. Voici quelques exemples actuels :

- Développement avec Amazon DynamoDB : conçu AWS par pour vous permettre de passer du statut de débutant à celui d'expert dans le développement d'applications réelles avec la modélisation des données pour Amazon DynamoDB.
- Cours d'approfondissement DynamoDB : cours créé par Pluralsight.
- Amazon DynamoDB : création d'applications NoSQL pilotées par des bases de données — Un cours organisé par l'équipe de formation et de certification sur edX.AWS

Lectures et écritures DynamoDB

Les lectures et les écritures DynamoDB font référence aux opérations qui extraient des données d'une table (lectures) et insèrent, mettent à jour ou suppriment des données dans une table (écritures). Lorsque vous utilisez DynamoDB, il est essentiel de comprendre les concepts de lecture et d'écriture, car ils ont un impact direct sur les performances et le coût de votre application.

Cette rubrique fournit des informations détaillées sur les différents types de cohérence en lecture qui s'appliquent à DynamoDB. Cette rubrique décrit également la consommation unitaire pour les différentes opérations de lecture et d'écriture que vous pouvez effectuer.

Rubriques

- Cohérence en lecture DynamoDB
- Opérations de lecture et d'écriture DynamoDB

Cohérence en lecture DynamoDB

Amazon DynamoDB lit les données à partir de tables, d'index secondaires locaux (LSI), d'index secondaires globaux (GSI) et de flux. Pour en savoir plus, consultez Composants de base d'Amazon DynamoDB. Les tables et les LSI proposent deux options de cohérence de lecture : des lectures éventuellement cohérentes (par défaut) et des lectures fortement cohérentes. Toutes les lectures provenant des index secondaires globaux (GSI) et des flux sont éventuellement cohérentes.

Lorsque votre application écrit des données dans une table DynamoDB et reçoit une réponse HTTP 200 (OK), cela signifie que l'écriture a eu lieu et qu'elle a été conservée durablement. DynamoDB fournit un isolement validé en lecture et assure que les opérations de lecture renvoient toujours des valeurs validées pour un élément. La lecture ne présente jamais un aperçu de l'élément issu d'une écriture qui n'a finalement pas aboutie. Un isolement validé en lecture n'empêche pas des modifications de l'élément juste après la lecture.

Lectures cohérentes à terme

Les lectures éventuellement cohérentes sont le modèle de cohérence de lecture par défaut pour toutes les opérations de lecture. Lorsque vous émettez des lectures éventuellement cohérentes d'une table DynamoDB ou d'un index, il se peut que les réponses ne reflètent pas les résultats d'une opération d'écriture récemment terminée. Si vous répétez votre demande de lecture

après un bref instant, la réponse doit finir par retourner l'élément le plus récent. Les lectures éventuellement cohérentes sont prises en charge sur les tables, les index secondaires locaux et les index secondaires globaux. Notez également que toutes les lectures d'un flux DynamoDB sont éventuellement cohérentes.

Les lectures éventuellement cohérentes coûtent deux fois moins cher que les lectures fortement cohérentes. Pour plus d'informations, consultez Tarification Amazon DynamoDB.

Lectures fortement cohérentes

Les opérations de lecture, telles que `GetItem`, `Query` et `Scan`, fournissent un paramètre `ConsistentRead` facultatif. Si vous définissez `ConsistentRead` sur true, DynamoDB retourne une réponse comportant les données les plus récentes, reflétant les mises à jour de toutes les opérations d'écriture précédentes qui ont réussi. Les lectures fortement cohérentes sont uniquement prises en charge dans les tables et les index secondaires locaux. Les lectures fortement cohérentes à partir d'un index secondaire global ou d'un flux DynamoDB ne sont pas prises en charge.

Cohérence de lecture des tables globales

DynamoDB prend également en charge les tables globales pour la réplication multiactive et à plusieurs régions. Une table globale est composée de plusieurs tables de réplicas dans différentes régions AWS. Toute modification apportée à un élément d'une table de réplica est répliquée dans tous les autres réplicas au sein de la même table globale, généralement en une seconde et elle devient éventuellement cohérente. Pour en savoir plus, consultez Modes de cohérence.

Opérations de lecture et d'écriture DynamoDB

Les opérations de lecture DynamoDB vous permettent d'extraire un ou plusieurs éléments d'une table en spécifiant la valeur de la clé de partition et, éventuellement, la valeur de la clé de tri. Les opérations d'écriture DynamoDB vous permettent d'insérer, de mettre à jour ou de supprimer des éléments dans une table. Cette rubrique explique la consommation d'unités de capacité pour ces deux opérations.

Rubriques

- Consommation des unités de capacité pour les opérations de lecture
- Consommation d'unités de capacité pour les opérations d'écriture

Consommation des unités de capacité pour les opérations de lecture

Les demandes de lecture de DynamoDB peuvent être soit fortement cohérentes, soit éventuellement cohérentes, soit transactionnelles.

- Une demande de lecture fortement cohérente d'un élément pouvant représenter jusqu'à 4 Ko nécessite une unité de lecture.
- Une demande de lecture cohérente à terme d'un élément pouvant représenter jusqu'à 4 Ko nécessite une demi-unité de lecture.
- Une demande de lecture transactionnelle d'un élément pouvant représenter jusqu'à 4 Ko nécessite deux unités de lecture.

Pour en savoir plus sur les modèles de cohérence de lecture dans DynamoDB, consultez Cohérence en lecture DynamoDB.

Les tailles d'élément pour les lectures sont arrondies au multiple de 4 Ko supérieur. Par exemple, la lecture d'un élément d'une taille de 3 500 octets consomme le même débit que la lecture d'un élément d'une taille de 4 Ko.

Si vous devez lire un élément d'une taille supérieure à 4 Ko, DynamoDB a besoin d'unités de lecture supplémentaires. Le nombre total d'unités de lecture nécessaires dépend de la taille de l'élément et du fait que vous vouliez ou pas une lecture cohérente à terme ou une lecture fortement cohérente. Par exemple, si la taille de votre élément est de 8 Ko, il vous faut 2 unités de lecture pour assurer une lecture fortement cohérente. Vous aurez besoin d'1 unité de lecture si vous choisissez des lectures cohérentes à terme ou de 4 unités de lecture pour une demande de lecture transactionnelle.

La liste suivante décrit comment les opérations de lecture DynamoDB consomment des unités de lecture :

- GetItem : lit un élément unique d'une table. Pour déterminer le nombre d'unités de lecture que l'opération `GetItem` consommera, arrondissez la taille de l'élément au multiple de 4 Ko supérieur. Il s'agit du nombre d'unités de lecture nécessaires, si vous avez spécifié une lecture fortement cohérente. Pour une lecture cohérente à terme, qui est la valeur par défaut, divisez ce nombre par deux.

  Par exemple, si vous lisez un élément d'une taille de 3,5 Ko, mise à l'échelle arrondit sa taille à 4 Ko. Si vous lisez un élément de 10 Ko, DynamoDB arrondit sa taille à 12 Ko.

- BatchGetItem : lit jusqu'à 100 éléments à partir d'une ou de plusieurs tables. DynamoDB traite chaque élément du lot comme une demande `GetItem` individuelle. DynamoDB arrondit d'abord la taille de chaque élément à la limite de 4 Ko suivante, puis calcule la taille totale. Le résultat n'est pas nécessairement identique à la taille totale de tous les éléments. Par exemple, si `BatchGetItem` lit deux éléments de tailles de 1,5 Ko et 6,5 Ko, la taille que DynamoDB calcule est de 12 Ko (4 Ko + 8 Ko). DynamoDB ne calcule pas la taille comme 8 Ko (1,5 Ko + 6,5 Ko).
- Query : lit plusieurs éléments ayant la même valeur de clé de partition. Tous les éléments renvoyés sont traités comme une seule opération de lecture, dans laquelle DynamoDB calcule la taille totale de tous les éléments. DynamoDB arrondit ensuite la taille à la limite de 4 Ko suivante. Par exemple, supposons que votre requête renvoie 10 éléments dont la taille combinée est de 40,8 Ko. DynamoDB arrondit la taille d'élément pour l'opération à 44 Ko. Si une requête renvoie 1 500 éléments de 64 octets chacun, la taille cumulée est de 96 Ko.
- Scan : lit tous les éléments d'une table. DynamoDB prend en compte la taille des éléments évalués, pas celle des éléments renvoyés par l'analyse. Pour plus d'informations sur les opérations d'analyse, consultez Analyse de tables dans DynamoDB.

> **Important**
>
> Si vous effectuez une opération de lecture sur un élément qui n'existe pas, DynamoDB consomme toujours le débit de lecture comme indiqué ci-dessus. Pour les opérations `Query`/`Scan`, un débit de lecture supplémentaire vous sera toujours facturé en fonction de la cohérence en lecture et du nombre de partitions recherchées pour répondre à la demande, même en l'absence de données.

Pour toute opération qui renvoie des éléments, vous pouvez demander un sous-ensemble d'attributs à extraire. Cependant, agir ainsi n'a aucun impct sur les calculs de taille d'élément. En outre, `Query` et `Scan` peuvent renvoyer des nombres d'éléments au lieu de valeurs d'attribut. L'obtention du nombre d'éléments utilise le même nombre d'unités de lecture et est soumise aux mêmes calculs de taille des éléments. En effet, DynamoDB doit lire chacun des éléments pour incrémenter le nombre.

Consommation d'unités de capacité pour les opérations d'écriture

Une unité d'écriture représente une écriture d'un élément d'une taille pouvant atteindre 1 Ko. Si vous avez besoin d'écrire un élément d'une taille supérieure à 1 Ko, DynamoDB doit consommer des unités d'écriture supplémentaires. Les demandes d'écriture transactionnelles requièrent 2 unités

d'écriture pour effectuer une écriture d'éléments d'une taille pouvant atteindre 1 Ko. Le nombre total d'unités requises de demande d'écriture dépend de la taille de l'élément. Par exemple, si la taille de votre élément est de 2 Ko, il vous faut 2 unités d'écriture pour assurer une demande d'écriture ou 4 unités d'écriture pour une demande d'écriture transactionnelle.

Les tailles d'élément pour les écritures sont arrondies au multiple de 1 Ko supérieur. Par exemple, l'écriture d'un élément de 500 octets consomme le même débit que l'écriture d'un élément de 1Ko.

La liste suivante décrit comment les opérations d'écriture DynamoDB consomment des unités d'écriture :

- PutItem : écrit un seul élément dans une table. Si un élément avec la même clé primaire existe dans la table, l'opération remplace l'élément. Pour calculer la consommation de débit approvisionné, la taille d'élément qui compte est la plus grande des deux.
- UpdateItem : modifie un élément unique dans la table. DynamoDB prend en compte la taille de l'élément tel qu'il apparaît avant et après la mise à jour. Le débit approvisionné consommé reflète la plus grande de ces tailles d'élément. Même si vous mettez à jour un sous-ensemble d'attributs de l'élément, `UpdateItem` utilise toujours la totalité du volume de débit approvisionné (la plus importante des tailles d'élément « avant » et « après »).
- DeleteItem : supprime un élément unique d'une table. La consommation de débit approvisionné est basée sur la taille de l'élément supprimé.
- BatchWriteItem : écrit jusqu'à 25 éléments dans une ou plusieurs tables. DynamoDB traite chaque élément du lot comme une demande `PutItem` ou `DeleteItem` individuelle (les mises à jour ne sont pas prises en charge). DynamoDB arrondit d'abord la taille de chaque élément à la limite de 1 Ko suivante, puis calcule la taille totale. Le résultat n'est pas nécessairement identique à la taille totale de tous les éléments. Par exemple, si `BatchWriteItem` écrit deux éléments de taille de 500 octets et 3,5 Ko, la taille que DynamoDB calcule est de 5 Ko (1 Ko + 4 Ko). DynamoDB ne calcule pas la taille comme 4 Ko (500 Ko + 3,5 Ko).

Pour les opérations `PutItem`, `UpdateItem`, et `DeleteItem`, DynamoDB arrondit la taille de l'élément à la limite de 1 Ko supérieure. Par exemple, si vous insérez ou supprimez un élément de 1,6 Ko, DynamoDB arrondit la taille de l'élément à 2 Ko.

`PutItem`, `UpdateItem` et `DeleteItem` permettent des écritures conditionnelles, dans lesquelles vous spécifiez une expression qui doit évaluer la valeur true pour que l'opération réussisse. Si le résultat de l'expression est false, DynamoDB continue à consommer des unités de capacité d'écriture de la table. Le nombre d'unités de capacité d'écriture utilisées dépend de la taille de l'élément. Cet

élément peut être un élément existant dans la table ou un nouvel élément que vous essayez de créer ou de mettre à jour. Supposons, par exemple, que la taille d'un élément existant est égale à 300 Ko. La taille du nouvel élément que vous essayez de créer ou de mettre à jour est égale à 310 Ko. La taille des unités de capacité d'écriture consommées sera égale à 310 Ko pour le nouvel article.

Capacité de débit DynamoDB

Cette section fournit une vue d'ensemble des deux modes de débit disponibles pour votre table DynamoDB et explique comment sélectionner le mode de capacité approprié pour votre application. Le mode de débit d'une table détermine la façon dont la capacité de celle-ci est gérée. Il détermine également le mode de facturation des opérations de lecture et d'écriture sur vos tables. Dans Amazon DynamoDB, vous avez le choix entre le mode à la demande et le mode provisionné pour vos tables, afin de répondre aux différentes exigences de charge de travail.

Rubriques

- Mode de capacité à la demande
- Mode alloué
- Mode de capacité à la demande DynamoDB
- Mode de capacité provisionnée DynamoDB
- Présentation du débit chaud DynamoDB
- Débordement et capacité adaptative DynamoDB
- Considérations relatives au changement de mode de capacité dans DynamoDB

Mode de capacité à la demande

Le mode à la demande Amazon DynamoDB est une option de débit sans serveur qui simplifie la gestion des bases de données et qui effectue automatiquement une mise à l'échelle pour prendre en charge les applications les plus exigeantes des clients. DynamoDB à la demande vous permet de créer une table sans vous soucier de la planification de la capacité, de la surveillance de l'utilisation ni de la configuration des politiques de mise à l'échelle. DynamoDB à la demande pay-per-request propose des tarifs pour les demandes de lecture et d'écriture afin que vous ne payiez que pour ce que vous utilisez. Pour les tables en mode à la demande, vous devez spécifier le débit de lecture et d'écriture que votre application est supposée atteindre.

Le mode à la demande est l'option de débit par défaut et recommandée pour la plupart des charges de travail DynamoDB. DynamoDB gère tous les aspects liés à la gestion et à la mise à l'échelle du débit, afin de prendre en charge des charges de travail qui peuvent commencer à petite échelle et atteindre des millions de demandes par seconde. Vous pouvez lire et écrire dans vos tables DynamoDB selon vos besoins, sans avoir à gérer la capacité de débit de la table. Pour de plus amples informations, veuillez consulter Mode de capacité à la demande DynamoDB.

Mode alloué

En mode provisionné, vous spécifiez le nombre de lectures et d'écritures par seconde nécessaires pour votre application. Vous serez facturé en fonction de la capacité horaire de lecture et d'écriture que vous avez provisionnée, et non de la quantité de cette capacité provisionnée que vous avez réellement consommée. Cela vous aide à maîtriser l'utilisation de DynamoDB de manière à ne pas dépasser un taux de demandes défini et de maintenir ainsi la prévisibilité des coûts.

Vous pouvez choisir d'utiliser la capacité provisionnée si vous avez des charges de travail stables avec une croissance prévisible, et si vous pouvez prévoir de manière fiable les besoins en capacité de votre application. Pour de plus amples informations, veuillez consulter Mode de capacité provisionnée DynamoDB.

Mode de capacité à la demande DynamoDB

Amazon DynamoDB à la demande offre une véritable expérience de base de données sans serveur, qui évolue automatiquement pour s'adapter aux charges de travail les plus exigeantes sans planification des capacités. Le mode à la demande simplifie le processus de configuration, élimine la gestion et la surveillance des capacités, et permet d'effectuer une mise à l'échelle rapide et automatique. En ce qui concerne la pay-per-request tarification, vous n'avez pas à vous soucier de la capacité inutilisée, car vous ne payez que pour le débit que vous utilisez réellement. Vous êtes facturé par demande de lecture ou d'écriture, de sorte que vos coûts reflètent directement votre utilisation réelle.

Lorsque vous choisissez le mode à la demande, DynamoDB s'adapte instantanément à vos charges de travail à mesure qu'elles augmentent ou diminuent jusqu'à tout niveau de trafic précédemment atteint. Si le niveau de trafic d'une charge de travail atteint un nouveau pic, DynamoDB est mis à l'échelle automatiquement pour prendre en charge les exigences de débit accrues. Le mode à la demande est l'option de débit par défaut que nous recommandons. Il simplifie la création d'applications modernes sans serveur capables de démarrer à petite échelle et permet de les mettre à l'échelle pour atteindre des millions de demandes par seconde. Une fois que votre table à la demande a été mise à l'échelle, vous pourrez instantanément atteindre à nouveau le même débit à l'avenir, sans limitation. Si vous ne générez aucun trafic vers votre table, aucun débit ne vous sera facturé avec le service à la demande. Pour plus d'informations sur les propriétés de mise à l'échelle du mode à la demande, consultez Débit initial et propriétés de mise à l'échelle.

Les tables qui utilisent le mode à la demande offrent les mêmes avantages, en matière de latence inférieure à 10 millisecondes, de contrat de niveau de service (SLA) et de sécurité, que les offres en mode provisionné de DynamoDB.

> **Note**
>
> Par défaut, DynamoDB vous protège contre toute utilisation involontaire et incontrôlée. Pour effectuer une mise à l'échelle au-delà des limites de débit de lecture et d'écriture de 40 000 au niveau des tables, pour toutes les tables de votre compte, vous pouvez demander une augmentation de ce quota. Les demandes de débit qui dépassent le quota de débit de la table par défaut sont limitées. Pour de plus amples informations, veuillez consulter Quotas de débit par défaut.

Vous avez également l'option de pouvoir configurer le débit maximal de lecture ou d'écriture (ou les deux) par seconde pour les tables individuelles à la demande et les index secondaires globaux. En configurant le débit, vous pouvez limiter l'utilisation et les coûts au niveau de la table, vous protéger contre une augmentation involontaire des ressources consommées et empêcher une utilisation excessive, pour bénéficier d'une gestion prévisible des coûts. Les demandes de débit qui dépassent le débit maximal de la table sont limitées. Vous pouvez modifier le débit maximal spécifique à la table à tout moment, en fonction des exigences de votre application. Pour de plus amples informations, veuillez consulter Débit maximal DynamoDB pour les tables à la demande.

Pour commencer, créez ou mettez à jour une table pour utiliser le mode à la demande. Pour de plus amples informations, veuillez consulter Opérations de base sur les tables DynamoDB.

Vous pouvez faire passer les tables du mode de capacité provisionnée au mode à la demande jusqu'à quatre fois par période de 24 heures. Vous pouvez à tout moment faire passer des tables du mode à la demande au mode de capacité provisionnée.

Pour plus d'informations sur le basculement entre les modes de capacité de lecture et d'écriture, consultez Considérations relatives au changement de mode de capacité dans DynamoDB. Pour connaître les quotas de table à la demande, consultez Débit de lecture/écriture.

Rubriques

- Unités de demande de lecture et unités de demande d'écriture
- Débit initial et propriétés de mise à l'échelle
- Débit maximal DynamoDB pour les tables à la demande

Unités de demande de lecture et unités de demande d'écriture

DynamoDB facture les lectures et écritures que votre application effectue dans vos tables, comptabilisées en termes d'unités de demande de lecture et d'unités de demande d'écriture.

Une unité de demande de lecture équivaut à une opération de lecture fortement cohérente par seconde, ou à deux opérations de lecture cohérente à terme par seconde, pour un élément dont la taille peut atteindre 4 Ko. Pour plus d'informations sur les modèles de cohérence de lecture dans DynamoDB, consultez Cohérence en lecture DynamoDB.

Une unité de demande d'écriture équivaut à une opération d'écriture par seconde, pour un élément d'une taille pouvant atteindre 1 Ko.

Pour plus d'informations sur la façon dont les unités de lecture et d'écriture sont consommées, consultez Opérations de lecture et d'écriture DynamoDB.

Débit initial et propriétés de mise à l'échelle

Les tables DynamoDB qui utilisent le mode de capacité à la demande s'adaptent automatiquement au volume de trafic de votre application. Les nouvelles tables à la demande pourront prendre en charge jusqu'à 4 000 écritures par seconde et 12 000 lectures par seconde. Le mode de capacité à la demande peut gérer jusqu'à deux fois le trafic de pointe précédent d'une table. Par exemple, imaginons que le modèle de trafic de votre application varie entre 25 000 et 50 000 lectures fortement cohérentes par seconde. 50 000 lectures par seconde correspondent au pic de trafic précédent. Le mode de capacité à la demande prend instantanément en charge un trafic soutenu pouvant atteindre 100 000 lectures par seconde. Si votre application connaît un trafic de 100 000 lectures par seconde, ce pic devient votre nouveau pic précédent. Ce pic précédent permet au trafic suivant d'atteindre jusqu'à 200 000 lectures par seconde.

Si votre charge de travail génère plus du double de votre pic précédent sur une table, DynamoDB alloue automatiquement plus de capacité à mesure que votre volume de trafic augmente. Cette allocation de capacité permet de garantir que votre charge de travail n'est pas limitée. Cette restriction peut cependant se produire si vous dépassez le double de votre trafic de pointe précédent dans les 30 minutes. Par exemple, imaginons que le modèle de trafic de votre application varie entre 25 000 et 50 000 lectures fortement cohérentes par seconde. 50 000 lectures par seconde correspondent au pic de trafic précédemment atteint. Nous vous recommandons de préchauffer votre table ou d'espacer la croissance du trafic sur au moins 30 minutes avant de générer plus de 100 000 lectures par seconde. Pour plus d'informations sur le préchauffage, consultez Présentation du débit chaud DynamoDB.

DynamoDB n'impose pas la restriction de limitation de 30 minutes si le pic de trafic de votre charge de travail reste inférieur au double du pic précédent. Si votre pic de trafic dépasse le double du pic, assurez-vous que cette croissance a lieu 30 minutes après votre dernier pic.

Débit maximal DynamoDB pour les tables à la demande

Pour les tables à la demande, vous pouvez éventuellement spécifier le débit maximal de lecture ou d'écriture (ou les deux) par seconde pour les tables individuelles et les index secondaires globaux associés ()GSIs. La spécification d'un débit maximal à la demande permet de limiter l'utilisation et les coûts au niveau de la table. Par défaut, les paramètres de débit maximal ne s'appliquent pas et votre débit à la demande est limité par un quota de service AWS de débit de lecture et d'écriture de 40 000 au niveau des tables, pour toutes les tables du compte. Si nécessaire, vous pouvez demander une augmentation de votre quota de service.

Lorsque vous configurez le débit maximal pour une table à la demande, les demandes de débit dépassant le montant maximum spécifié seront limitées. Vous pouvez modifier les paramètres de débit au niveau de la table à tout moment, en fonction des exigences de votre application.

Voici quelques cas d'utilisation courants qui peuvent bénéficier d'une utilisation d'un débit maximal pour les tables à la demande :

- Optimisation des coûts de débit : l'utilisation d'un débit maximal pour les tables à la demande fournit une couche supplémentaire de prévisibilité et de gérabilité des coûts. En outre, il offre plus de flexibilité pour utiliser le mode à la demande afin de prendre en charge des charges de travail présentant des modèles de trafic et des budgets différents.
- Protection contre une utilisation excessive : en définissant un débit maximal, vous pouvez éviter une augmentation accidentelle de la consommation en lecture ou en écriture, qui pourrait résulter d'un code non optimisé ou de processus malveillants, par rapport à une table à la demande. Ce paramètre au niveau de la table peut éviter aux organisations de consommer trop de ressources dans un certain laps de temps.
- Protection des services en aval : une application client peut inclure des technologies avec et sans serveur. La partie sans serveur de l'architecture peut évoluer rapidement pour répondre à la demande. Toutefois, les composants en aval dotés de capacités fixes pourraient être débordés. La mise en œuvre de paramètres de débit maximal pour les tables à la demande peut empêcher la propagation d'un grand nombre d'événements vers plusieurs composants en aval, avec des effets secondaires inattendus.

Vous pouvez configurer le débit maximal pour le mode à la demande pour les tables mono-régionales et les tables globales nouvelles et existantes et. GSIs Vous pouvez également configurer le débit maximal lors de la restauration des tables et de l'importation de données à partir des flux de travail Amazon S3.

Vous pouvez définir les paramètres de débit maximal pour une table à la demande à l'aide de la console DynamoDB, de l'AWS CLI, d'AWS CloudFormation ou de l'API DynamoDB.

> **Note**
>
> Le débit maximal pour une table à la demande est appliqué dans la mesure du possible et doit être considéré comme des objectifs plutôt que comme des plafonds de demandes garantis. Votre charge de travail peut temporairement dépasser le débit maximal spécifié en raison de la capacité de débordement. Dans certains cas, DynamoDB consomme la capacité de débordement pour gérer les lectures ou écritures excédentaires par rapport aux paramètres de débit maximal de votre table. La capacité de transmission en mode rafale permet de satisfaire les demandes inattendues en lecture ou en écriture, alors qu'elles seraient sinon limitées.

Rubriques

- Considérations relatives à l'utilisation du débit maximal pour le mode à la demande
- Limitation des demandes et mesures CloudWatch

Considérations relatives à l'utilisation du débit maximal pour le mode à la demande

Lorsque vous utilisez le débit maximal pour les tables en mode à la demande, les considérations suivantes s'appliquent :

- Vous pouvez définir indépendamment le débit maximal applicable aux lectures et aux écritures pour n'importe quelle table à la demande ou définir un index secondaire global individuel au sein de cette table afin d'optimiser votre approche en fonction d'exigences spécifiques.
- Vous pouvez utiliser Amazon CloudWatch pour surveiller et comprendre les mesures d'utilisation de DynamoDB au niveau des tables et pour déterminer les paramètres de débit maximal appropriés pour le mode à la demande. Pour de plus amples informations, veuillez consulter Métriques et dimensions DynamoDB.

- Lorsque vous spécifiez les paramètres de débit maximal en lecture ou en écriture (ou les deux) sur un réplica de table globale, les mêmes paramètres de débit maximal sont automatiquement appliqués à toutes les tables de réplica. Pour garantir une réplication correcte des données, il est important que les tables de réplica et les index secondaires dans votre table globale aient des paramètres de débit d'écriture identiques. Pour de plus amples informations, veuillez consulter Bonnes pratiques relatives aux tables globales.
- Le plus petit débit maximal de lecture ou d'écriture que vous pouvez spécifier est d'une unité de demande par seconde.
- Le débit maximal que vous spécifiez doit être inférieur au quota de débit par défaut disponible pour toute table à la demande ou pour un index secondaire global individuel au sein de cette table.

Limitation des demandes et mesures CloudWatch

Si votre application dépasse le débit de lecture ou d'écriture maximal que vous avez défini dans votre table à la demande, DynamoDB commence à limiter ces demandes. Quand DynamoDB limite une opération de lecture ou d'écriture, un code `ThrottlingException` est renvoyé à l'appelant. Vous pouvez ensuite prendre les mesures appropriées, le cas échéant. Par exemple, vous pouvez augmenter ou désactiver le paramètre de débit maximal des tables, ou attendre quelques instants avant de réessayer.

Pour simplifier la surveillance du débit maximal configuré pour une table ou un index secondaire global, CloudWatch fournit les mesures suivantes : OnDemandMaxReadRequestUnits etOnDemandMaxWriteRequestUnits.

Mode de capacité provisionnée DynamoDB

Lorsque vous créez une table approvisionnée dans DynamoDB, vous devez spécifier sa capacité de débit provisionné. Il s'agit du volume de débit de lecture et d'écriture que la table peut prendre en charge. Vous serez facturé en fonction de la capacité horaire de lecture et d'écriture que vous avez provisionnée, et non de la quantité de cette capacité provisionnée que vous avez réellement consommée.

À mesure que les exigences de données et d'accès de votre application changent, vous pouvez avoir besoin d'ajuster les paramètres de débit de votre table. Vous pouvez recourir à la scalabilité automatique pour ajuster automatiquement la capacité allouée de votre table en fonction de l'évolution du trafic. L'autoscaling de DynamoDB utilise une politique de mise à l'échelle dans Application Auto Scaling. Pour configurer l'autoscaling dans DynamoDB, définissez les niveaux

minimum et maximum de capacité de lecture et d'écriture en plus du pourcentage d'utilisation cible. Application Auto Scaling crée et gère les CloudWatch alarmes qui déclenchent des événements de dimensionnement lorsque la métrique s'écarte de la cible. L'autoscaling surveille l'activité de votre table et ajuste ses paramètres de capacité à la hausse ou à la baisse en fonction de seuils préconfigurés. La mise à l'échelle automatique se déclenche lorsque la capacité consommée dépasse l'utilisation cible configurée pendant deux minutes consécutives. CloudWatch les alarmes peuvent avoir un court délai de quelques minutes avant de déclencher le redimensionnement automatique. Pour de plus amples informations, veuillez consulter Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

Si vous utilisez la scalabilité automatique de DynamoDB, les paramètres de débit sont automatiquement ajustés en fonction des charges de travail réelles. Vous pouvez aussi utiliser l'opération UpdateTable pour ajuster manuellement la capacité de débit de votre table. Par exemple, vous pouvez décider de procéder de la sorte si vous devez charger en masse des données à partir d'un magasin de données existant dans votre nouvelle table DynamoDB. Vous pouvez créer la table avec un paramètre de débit en écriture élevé, puis diminuer ce paramètre une fois que le chargement en masse des données est terminé.

> **Note**
>
> Par défaut, DynamoDB vous protège contre toute utilisation involontaire et incontrôlée. Pour effectuer une mise à l'échelle au-delà des limites de débit de lecture et d'écriture de 40 000 au niveau des tables, pour toutes les tables de votre compte, vous pouvez demander une augmentation de ce quota. Les demandes de débit qui dépassent le quota de débit de la table par défaut sont limitées. Pour de plus amples informations, veuillez consulter Quotas de débit par défaut.

Vous pouvez faire passer les tables du mode de capacité provisionnée au mode à la demande jusqu'à quatre fois par période de 24 heures. Vous pouvez à tout moment faire passer des tables du mode à la demande au mode de capacité provisionnée.

Pour plus d'informations sur le basculement entre les modes de capacité de lecture et d'écriture, consultez Considérations relatives au changement de mode de capacité dans DynamoDB.

Rubriques

- Unités de capacité en lecture et unités de capacité en écriture
- Choix des paramètres de débit initial

- Mise à l'échelle automatique de DynamoDB
- Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB
- Capacité réservée DynamoDB

Unités de capacité en lecture et unités de capacité en écriture

Pour les tables du mode provisionné, vous spécifiez les exigences de débit en matière d'unités de capacité. Ces unités représentent la quantité de données que votre application doit lire ou écrire par seconde. Si nécessaire, vous pouvez modifier ces paramètres ultérieurement, ou activer la scalabilité automatique de DynamoDB.

Pour un élément dont la taille peut atteindre 4 Ko, une unité de capacité en lecture équivaut à une opération de lecture fortement cohérente par seconde, ou à deux opérations de lecture cohérente à terme par seconde. Pour plus d'informations sur les modèles de cohérence de lecture dans DynamoDB, consultez Cohérence en lecture DynamoDB.

Une unité de capacité d'écriture équivaut à une écriture par seconde pour un élément d'une taille pouvant atteindre 1 Ko. Pour plus d'informations sur les différentes opérations d'écriture et de lecture, consultez Opérations de lecture et d'écriture DynamoDB.

Choix des paramètres de débit initial

Chaque application a des exigences spécifiques pour la lecture et l'écriture dans une base de données. Lorsque vous déterminez les paramètres de débit initial pour une table DynamoDB, prenez en considération les points suivants :

- Taux de demandes de lecture et d'écriture attendus : vous devez estimer le nombre de lectures et d'écritures que vous devez effectuer par seconde.
- Taille des éléments : certains éléments sont d'une taille suffisamment petite pour être lus ou écrits par une seule unité de capacité. Les éléments plus volumineux exigent plusieurs unités de capacité. En évaluant la taille moyenne des éléments qui figureront dans votre table, vous pouvez spécifier les paramètres de débit provisionné qui conviennent pour votre table.
- Exigences en matière de cohérence des lectures : les unités de capacité de lecture sont basées sur des opérations de lecture fortement cohérentes et utilisent deux fois plus de ressources de base de données que les lectures cohérentes à terme. Vous devez déterminer si votre application requière des lectures cohérentes fortes, ou si elle peut assouplir cette exigence et exécuter des lectures éventuellement cohérentes à la place. Par défaut, les opérations de lecture dans

DynamoDB sont cohérentes à terme. Si nécessaire, vous pouvez demander des lectures fortement cohérentes pour ces opérations.

Par exemple, imaginons que vous vouliez lire 80 éléments d'une table par seconde. Ces éléments ont une taille de 3 Ko et vous voulez des lectures fortement cohérentes. Dans ce cas, chaque lecture nécessite une unité de capacité de lecture provisionnée. Pour déterminer ce nombre, divisez la taille d'élément de l'opération par 4 Ko. Puis, arrondissez au nombre entier le plus proche, comme indiqué dans l'exemple suivant :

- 3 Ko / 4 Ko = 0,75, ou 1 unité de capacité de lecture

Par conséquent, pour lire 80 éléments par seconde à partir d'une table, définissez le débit de lecture provisionné de la table sur 80 unités de capacité de lecture, comme indiqué dans l'exemple suivant :

- 1 unité de capacité de lecture par élément x 80 lectures par seconde = 80 unités de capacité de lecture

Supposons à présent que vous vouliez écrire 100 éléments par seconde sur votre table et que chaque élément ait une taille de 512 octets. Dans ce cas, chaque écriture nécessite une unité de capacité d'écriture provisionnée. Pour déterminer ce nombre, divisez la taille d'élément de l'opération par 1 Ko. Puis, arrondissez au nombre entier le plus proche, comme indiqué dans l'exemple suivant :

- 512 octets/1 Ko = 0,5 ou 1 unité de capacité d'écriture

Pour écrire 100 éléments par seconde dans votre table, définissez le débit d'écriture provisionné de la table sur 100 unités de capacité d'écriture :

- 1 unité de capacité d'écriture par élément x 100 écritures par seconde = 100 unités de capacité d'écriture

Mise à l'échelle automatique de DynamoDB

L'autoscaling de DynamoDB gère activement la capacité de débit pour les tables et les index secondaires globaux. Avec la scalabilité automatique, vous définissez une plage (limites supérieure et inférieure) pour les unités de capacité en lecture et en écriture. Vous définissez également un

pourcentage d'utilisation cible dans cette plage. DynamoDB vise à maintenir votre utilisation cible, même lorsque la charge de travail de votre application augmente ou diminue.

Avec la scalabilité automatique de DynamoDB, une table ou un index secondaire global peut augmenter sa capacité de lecture et d'écriture approvisionnée de manière à gérer une augmentation soudaine du trafic, sans limitation de demande. Lorsque la charge de travail diminue, la scalabilité automatique de DynamoDB peut réduire le débit de sorte que vous ne payez pas pour une capacité approvisionnée non utilisée.

> **Note**
>
> Si vous utilisez le AWS Management Console pour créer une table ou un index secondaire global, le dimensionnement automatique de DynamoDB est activé par défaut.
> Vous pouvez gérer les paramètres de mise à l'échelle automatique à tout moment à l'aide de la console AWS CLI, du ou de l'un des AWS SDKs. Pour de plus amples informations, veuillez consulter Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

Taux d'utilisation

Le taux d'utilisation peut vous aider à déterminer si vous surchargez la capacité de provisionnement. Dans ce cas, vous devez réduire la capacité de votre table pour réduire les coûts. À l'inverse, cela peut également vous aider à déterminer si votre capacité de provisionnement est insuffisante. Dans ce cas, vous devez augmenter la capacité de la table pour éviter toute limitation potentielle des demandes lors d'instances imprévues à trafic élevé. Pour plus d'informations, consultez Amazon DynamoDB auto scaling: Performance and cost optimization at any scale.

Si vous utilisez l'autoscaling DynamoDB, vous devez également définir un pourcentage d'utilisation cible. L'autoscaling utilisera ce pourcentage comme cible pour ajuster la capacité à la hausse ou à la baisse. Nous recommandons de définir l'objectif d'utilisation sur 70 %. Pour de plus amples informations, veuillez consulter Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB

De nombreuses charges de travail de base de données sont cycliques par nature, tandis que d'autres sont difficiles à prévoir. Prenons l'exemple d'une application de réseau social où la plupart des

utilisateurs sont actifs pendant la journée. La base de données doit être capable de gérer l'activité durant cette période, mais elle n'a pas besoin des mêmes niveaux de débit pendant la nuit. Prenons également l'exemple d'une nouvelle application de jeux pour appareils mobiles qui, contre toute attente, est rapidement adoptée par les utilisateurs. Si le jeu devient trop populaire, les ressources de base de données disponibles risquent d'être dépassées, entraînant un ralentissement des performances et le mécontentement des clients. Ce type de charges de travail nécessite souvent une intervention manuelle pour augmenter ou diminuer les ressources de base de données en fonction de la variation des niveaux d'utilisation.

Amazon DynamoDB Auto Scaling AWS utilise le service Application Auto Scaling pour ajuster dynamiquement la capacité de débit allouée en votre nom, en réponse aux modèles de trafic réels. Cela permet à une table ou à un index secondaire global (GSI) d'augmenter sa capacité de lecture et d'écriture approvisionnée afin de gérer les hausses soudaines de trafic sans limitation. Lorsque la charge de travail diminue, la scalabilité automatique d'application réduit le débit de sorte que vous ne payez pas pour une capacité approvisionnée non utilisée.

> **Note**
>
> Si vous utilisez le AWS Management Console pour créer une table ou un index secondaire global, le dimensionnement automatique de DynamoDB est activé par défaut. Vous pouvez modifier vos paramètres de scalabilité automatique à tout moment. Pour de plus amples informations, veuillez consulter Utilisation de la mise AWS Management Consoleà l'échelle automatique avec DynamoDB.
>
> Lorsque vous supprimez une table ou une réplique de table globale, les cibles évolutives, les politiques de dimensionnement ou les CloudWatch alarmes associées ne sont pas automatiquement supprimées avec elle.

Avec la scalabilité automatique d'application, vous créez une politique de mise à l'échelle pour une table ou un index secondaire global. La politique de mise à l'échelle spécifie si vous souhaitez mettre à l'échelle la capacité de lecture ou d'écriture (ou les deux), ainsi que les paramètres d'unité de capacité approvisionnée minimum et maximum pour la table ou l'index.

La politique de mise à l'échelle contient également une utilisation cible, c'est-à-dire le pourcentage du débit approvisionné consommé à un moment donné. La scalabilité automatique d'application utilise un algorithme de suivi de cible pour ajuster le débit approvisionné de la table (ou de l'index) en réponse aux charges de travail réelles, de façon que l'utilisation de capacité réelle reste au niveau ou à un niveau proche de votre utilisation cible.

DynamoDB produit le débit provisionné consommé pendant des périodes d'une minute. La mise à l'échelle automatique se déclenche lorsque la capacité consommée dépasse l'utilisation cible configurée pendant deux minutes consécutives. CloudWatch les alarmes peuvent avoir un court délai de quelques minutes avant de déclencher le redimensionnement automatique. Ce délai garantit une évaluation CloudWatch précise des mesures. Si les pics de débit consommés sont espacés de plus d'une minute, l'autoscaling risque de ne pas se déclencher. De même, un événement de réduction verticale peut être déclenché lorsque 15 points de données consécutifs sont en dessous de l'utilisation cible. Dans les deux cas, après le déclenchement de l'autoscaling, l'UpdateTableAPI est invoquée. Plusieurs minutes sont ensuite nécessaires pour mettre à jour la capacité provisionnée pour la table ou l'index. Pendant cette période, toutes les demandes dépassant la capacité provisionnée précédente pour les tables seront limitées.

> **Important**
>
> Vous ne pouvez pas ajuster le nombre de points de données à dépasser pour déclencher l'alarme sous-jacente (bien que le nombre actuel puisse changer à l'avenir).

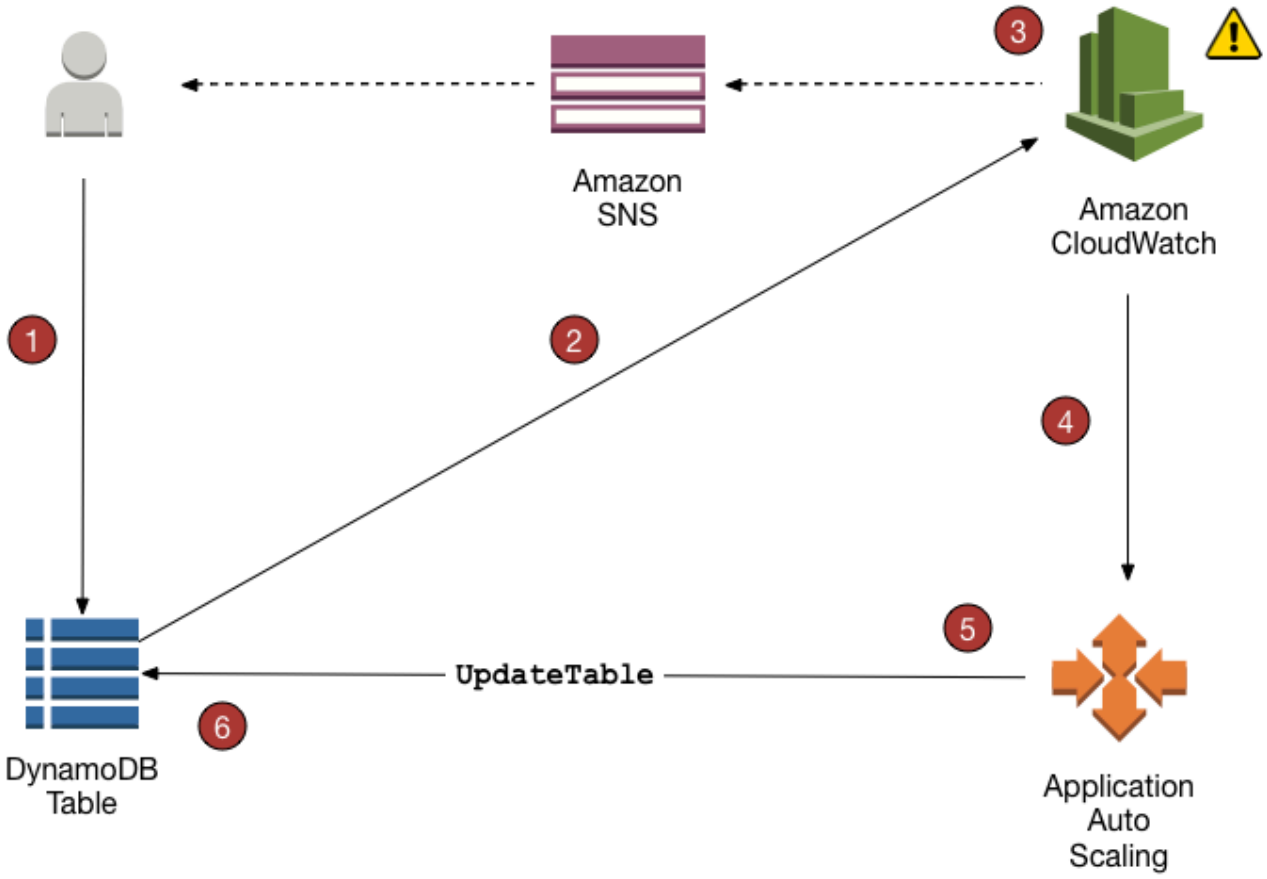
Vous pouvez également définir les valeurs d'utilisation cibles de la scalabilité automatique entre 20 % et 90 % pour votre capacité de lecture et d'écriture.

> **Note**
>
> Outre les tables, la scalabilité automatique de DynamoDB prend également en charge les index secondaires globaux. Chaque index secondaire global a sa propre capacité de débit approvisionné, distincte de celle de sa table de base. Lorsque vous créez une politique de mise à l'échelle pour un index secondaire global, la scalabilité automatique d'application ajuste les paramètres de débit approvisionné pour l'index afin de s'assurer que l'utilisation réelle reste au niveau ou à une niveau proche du ratio d'utilisation souhaité.

Comment fonctionne la scalabilité automatique de DynamoDB

> **Note**
>
> Pour commencer rapidement à utiliser la scalabilité automatique de DynamoDB, consultez Utilisation de la mise AWS Management Consoleà l'échelle automatique avec DynamoDB.

Le diagramme suivant fournit une présentation de haut niveau de la manière dont la scalabilité automatique de DynamoDB gère la capacité de débit pour une table.



Les étapes suivantes résument le processus de scalabilité automatique illustré dans le diagramme précédent :

1. Vous créez une politique de scalabilité automatique d'application pour votre table DynamoDB.
2. DynamoDB publie des statistiques de capacité consommée sur Amazon. CloudWatch
3. Si la capacité consommée de la table dépasse votre objectif d'utilisation (ou tombe en dessous de l'objectif) pendant un certain temps, Amazon CloudWatch déclenche une alarme. Vous pouvez voir l'alarme sur la console et recevoir des notifications via Amazon Simple Notification Service (Amazon SNS).
4. L' CloudWatch alarme appelle Application Auto Scaling pour évaluer votre politique de dimensionnement.
5. La scalabilité automatique d'application émet une demande `UpdateTable` pour ajuster le débit approvisionné de votre table.

6. DynamoDB traite la demande `UpdateTable` en augmentant (ou réduisant) de manière dynamique la capacité de débit approvisionné de la table de façon que celle-ci se rapproche de votre utilisation cible.

Pour comprendre la manière dont fonctionne la scalabilité automatique de DynamoDB, supposons que vous ayez une table nommée `ProductCatalog`. Cette table est chargée en masse de manière occasionnelle, elle n'entraîne donc pas une importante activité en écriture. Toutefois, elle ne connaît pas une activité en lecture très élevée, qui varie avec le temps. En surveillant les CloudWatch métriques Amazon pour`ProductCatalog`, vous déterminez que la table nécessite 1 200 unités de capacité de lecture (pour éviter que DynamoDB ne limite les demandes de lecture lorsque l'activité est maximale). Vous déterminez également que `ProductCatalog` requiert au minimum 150 unités de capacité de lecture lorsque le trafic de lecture est à son point le plus bas. Pour plus d'informations sur la prévention de la limitation, consultez Résolution des problèmes de limitation dans Amazon DynamoDB.

Dans la plage de 150 à 1 200 unités de capacité de lecture, vous décidez qu'une utilisation cible de 70 % conviendrait pour la table `ProductCatalog`. L'utilisation cible correspond au ratio des unités de capacité consommées par rapport aux unités de capacité approvisionnées, exprimé sous forme de pourcentage. La scalabilité automatique d'application utilise son algorithme de suivi de cible pour s'assurer que la capacité de lecture approvisionnée de `ProductCatalog` est correctement ajustée, de sorte que l'utilisation demeure proche de 70 %.

> **Note**
>
> La scalabilité automatique de DynamoDB modifie les paramètres de débit approvisionné uniquement lorsque la charge de travail réelle reste élevée ou basse pendant une période continue de plusieurs minutes. L'algorithme de suivi de cible de la scalabilité automatique d'application cherche à maintenir l'utilisation cible au niveau ou à un niveau proche de la valeur choisie sur le long terme.
>
> Les pics soudains de l'activité de lecture sont gérés par la capacité de transmission en mode rafale intégrée de la table. Pour de plus amples informations, veuillez consulter Capacité de débordement.

Pour activer la scalabilité automatique de DynamoDB pour la table `ProductCatalog`, vous devez créer une politique de mise à l'échelle. La politique spécifie les éléments suivants :

- Table ou index secondaire global que vous voulez gérer
- Le type de capacité à gérer (lecture ou écriture)
- Les limites supérieure et inférieure pour les paramètres du débit provisionné.
- Votre utilisation cible

Lorsque vous créez une politique de dimensionnement, Application Auto Scaling crée une paire d' CloudWatch alarmes Amazon en votre nom. Chaque paire représente les limites supérieure et inférieure pour vos paramètres de débit provisionné. Ces CloudWatch alarmes sont déclenchées lorsque l'utilisation réelle de la table s'écarte de votre utilisation cible pendant une période prolongée.

Lorsque l'une des CloudWatch alarmes est déclenchée, Amazon SNS vous envoie une notification (si vous l'avez activée). L' CloudWatch alarme appelle ensuite Application Auto Scaling, qui à son tour indique à DynamoDB d'ajuster la capacité allouée à la `ProductCatalog` table à la hausse ou à la baisse, selon le cas.

Lors d'un événement de dimensionnement,AWS Config est facturé par élément de configuration enregistré. Lorsqu'un événement de dimensionnement se produit, quatre CloudWatch alarmes sont créées pour chaque événement d'auto-scaling en lecture et en écriture : ProvisionedCapacity alarmes : ProvisionedCapacityLow, ProvisionedCapacityHigh et ConsumedCapacity alarmes : AlarmHigh,. AlarmLow Cela se traduit par un total de huit alarmes. Par conséquent,AWS Config enregistre huit éléments de configuration pour chaque événement de dimensionnement.

> **Note**
>
> Vous pouvez également planifier la mise à l’échelle de DynamoDB de manière à ce qu’elle se produise à certains moments. Découvrez les étapes de base ici.

Notes d’utilisation

Avant de commencer à utiliser la scalabilité automatique de DynamoDB, vous devez être conscient de ce qui suit :

- La scalabilité automatique de DynamoDB peut augmenter la capacité de lecture ou d’écriture aussi souvent que nécessaire, selon votre politique de scalabilité automatique. Tous les quotas DynamoDB restent en vigueur, comme décrit dans Quotas dans Amazon DynamoDB.

- La scalabilité automatique de DynamoDB ne vous empêche pas de modifier manuellement les paramètres de débit approvisionné. Ces réglages manuels n'ont aucune incidence sur les CloudWatch alarmes existantes liées au dimensionnement automatique de DynamoDB.
- Si vous activez la scalabilité automatique de DynamoDB pour une table contenant un ou plusieurs index secondaires globaux, nous vous recommandons vivement d'appliquer également la scalabilité automatique de manière uniforme à ces index. Cela permettra de garantir de meilleures performances pour les écritures et les lectures de table et d'éviter les limitations. Vous pouvez activer la mise à l'échelle automatique en sélectionnant Apply same settings to global secondary indexes (Appliquer les mêmes paramètres aux index secondaires globaux) dans la AWS Management Console. Pour de plus amples informations, veuillez consulter Activation de la scalabilité automatique de DynamoDB sur des tables existantes.
- Lorsque vous supprimez une table ou une réplique de table globale, les cibles évolutives, les politiques de dimensionnement ou les CloudWatch alarmes associées ne sont pas automatiquement supprimées.
- Lors de la création d'un index secondaire global pour une table existante, la scalabilité automatique n'est pas activée pour l'index secondaire global. Vous devrez gérer manuellement la capacité pendant la construction de l'index secondaire global. Une fois que le remplissage de l'index secondaire global est terminé et qu'il a atteint le statut actif, la mise à l'échelle automatique fonctionne normalement.

Utilisation de la mise AWS Management Consoleà l'échelle automatique avec DynamoDB

Lorsque vous utilisez le AWS Management Console pour créer une nouvelle table, le dimensionnement auto d'Amazon DynamoDB est activé par défaut pour cette table. Vous pouvez également utiliser la console pour activer la scalabilité automatique pour des tables existantes, modifier les paramètres de scalabilité automatique ou désactiver la scalabilité automatique.

> **Note**
>
> Pour des fonctionnalités plus avancées, telles que la définition des temps de recharge de mise à l'échelle initiale et inférieure, utilisez le AWS Command Line Interface() pour AWS CLI gérer le dimensionnement automatique de DynamoDB. Pour de plus amples informations, veuillez consulter Utilisation du AWS CLI pour gérer le dimensionnement automatique de DynamoDB.

Rubriques

- Avant de commencer : octroi d’autorisations utilisateur pour la scalabilité automatique de DynamoDB
- Création d’une table avec la scalabilité automatique activée
- Activation de la scalabilité automatique de DynamoDB sur des tables existantes
- Affichage des activités de scalabilité automatique sur la console
- Modification ou désactivation des paramètres de scalabilité automatique de DynamoDB

Avant de commencer : octroi d’autorisations utilisateur pour la scalabilité automatique de DynamoDB

Dans Gestion des identités et des accès AWS(IAM), la politique AWS gérée `DynamoDBFullAccess` fournit les autorisations requises pour utiliser la console DynamoDB. Toutefois, pour la scalabilité automatique de DynamoDB, les utilisateurs doivent disposer d’autorisations supplémentaires.

> **Important**
>
> Pour supprimer une table pour laquelle l’autoscaling est activé, des autorisations `application-autoscaling:*` sont nécessaires. La politique AWS gérée `DynamoDBFullAccess` inclut de telles autorisations.

Pour configurer un utilisateur pour l'accès à la console DynamoDB et le dimensionnement automatique de DynamoDB, créez un rôle et ajoutez la politique d'accès à ce rôle. AmazonDynamo DBFull Attribuez ensuite le rôle à un utilisateur.

Création d’une table avec la scalabilité automatique activée

> **Note**
>
> La scalabilité automatique de DynamoDB nécessite la présence d’un rôle lié à un service (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) qui effectue pour vous les actions de scalabilité automatique. Ce rôle est créé automatiquement pour vous. Pour plus d’informations, consultez Rôles liés à un service pour Application Auto Scaling dans le Guide de l’utilisateur Application Auto Scaling.

Pour créer une table avec la scalabilité automatique activée

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Choisissez Créer un tableau.
3. Sur la page Créer une table, saisissez le Nom de la table et les détails de la clé primaire.
4. Si vous choisissez Paramètres par défaut, l'autoscaling est activé dans la nouvelle table.

   Sinon, choisissez Personnaliser les paramètres et procédez comme suit pour définir les paramètres personnalisés de la table :

   a. Pour la Classe Table, conservez la sélection par défaut de DynamoDB Standard.

   b. Pour les Paramètres de capacité de lecture/écriture, conservez la sélection par défaut Provisionné, puis procédez comme suit :

      i. Pour la Capacité de lecture, assurez-vous que l'Autoscaling est défini sur Activé.

      ii. Pour la Capacité d'écriture, assurez-vous que l'Autoscaling est défini sur Activé.

      iii. Pour la Capacité de lecture et la Capacité d'écriture, définissez votre politique de mise à l'échelle souhaitée pour la table et, éventuellement, tous les index secondaires globaux de la table.

      - Unités de capacité minimum- Saisissez votre limite inférieure pour la plage de scalabilité automatique.
      - Unités de capacité maximum - Saisissez votre limite supérieure pour la plage de scalabilité automatique.
      - Utilisation cible - Saisissez votre pourcentage d'utilisation cible pour la table.

      > **Note**
      >
      > Si vous créez un index secondaire global pour la nouvelle table, la capacité de l'index au moment de la création sera la même que celle de votre table de base. Vous pouvez modifier la capacité de l'index dans les paramètres de la table après avoir créé la table.

5. Choisissez Créer un tableau. Cela crée votre table avec les paramètres d'autoscaling que vous avez spécifiés.

Activation de la scalabilité automatique de DynamoDB sur des tables existantes

> **Note**
>
> La scalabilité automatique de DynamoDB nécessite la présence d'un rôle lié à un service (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) qui effectue pour vous les actions de scalabilité automatique. Ce rôle est créé automatiquement pour vous. Pour plus d'informations, veuillez consulter Rôles liés à un service pour scalabilité automatique d'application.

Pour activer la scalabilité automatique de DynamoDB pour une table existante

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez la table sur laquelle vous souhaitez activer l'autoscaling, puis procédez comme suit :
   a. Choisissez l'onglet Paramètres supplémentaires.
   b. Dans la section Capacité de lecture/écriture, sélectionnez Modifier.
   c. Dans la section Mode de capacité, sélectionnez Alloué.
   d. Dans la section Table capacity (Capacité de la table), définissez Auto scaling (Scalabilité automatique) sur On (Activé) pour Read capacity (Capacité de lecture), Write capacity (Capacité d'écriture), ou les deux. Pour chacun de ces éléments, définissez votre politique de mise à l'échelle souhaitée pour la table et, éventuellement, tous les index secondaires globaux de la table.
      - Unités de capacité minimum- Saisissez votre limite inférieure pour la plage de scalabilité automatique.
      - Unités de capacité maximum - Saisissez votre limite supérieure pour la plage de scalabilité automatique.
      - Utilisation cible - Saisissez votre pourcentage d'utilisation cible pour la table.
      - Utilisez les mêmes paramètres de read/write capacité pour tous les index secondaires globaux : choisissez si les index secondaires globaux doivent utiliser la même politique de dimensionnement automatique que la table de base.

> **Note**
>
> Pour de meilleures performances, nous vous recommandons d'activer Utiliser les mêmes paramètres read/write de capacité pour tous les index secondaires globaux. Cette option permet à la scalabilité automatique de DynamoDB de mettre à l'échelle de manière uniforme tous les index secondaires globaux sur la table de base. Cela inclut les index secondaires globaux existants et tous les index que vous créerez ultérieurement pour cette table.
> Lorsque cette option est activée, vous ne pouvez pas définir de politique de mise à l'échelle sur un index secondaire global individuel.

4. Lorsque les paramètres vous conviennent, choisissez Enregistrer.

Affichage des activités de scalabilité automatique sur la console

A mesure que votre application génère du trafic de lecture et d'écriture vers votre table, la scalabilité automatique de DynamoDB modifie de manière dynamique les paramètres de débit de la table. Amazon CloudWatch assure le suivi de la capacité allouée et consommée, des événements limités, de la latence et d'autres indicateurs pour toutes vos tables DynamoDB et vos index secondaires.

Pour afficher ces métriques dans la console DynamoDB, choisissez la table avec laquelle vous souhaitez travailler et sélectionnez l'onglet Contrôler. Pour créer une vue personnalisable des statistiques du tableau, sélectionnez Afficher tout dans CloudWatch.

Modification ou désactivation des paramètres de scalabilité automatique de DynamoDB

Vous pouvez utiliser le AWS Management Console pour modifier les paramètres de mise à l'échelle automatique de DynamoDB. Pour ce faire, accédez à l'onglet Paramètres supplémentaires pour votre table et sélectionnez Modifier dans la section Capacité de lecture/écriture. Pour plus d'informations sur ces paramètres, consultez la page Activation de la scalabilité automatique de DynamoDB sur des tables existantes.

Utilisation du AWS CLI pour gérer le dimensionnement automatique de DynamoDB

Au lieu d'utiliser le ()AWS Management Console, vous pouvez utiliser le AWS Command Line Interface(AWS CLI) pour gérer le dimensionnement automatique d'Amazon DynamoDB. Le didacticiel dans cette section montre comment installer et configurer l'AWS CLI pour gérer la scalabilité automatique de DynamoDB. Dans ce didacticiel, vous allez effectuer les opérations suivantes :

- Créez une table DynamoDB nommée `TestTable`. Les paramètres de débit initial sont 5 unités de capacité de lecture et 5 unités de capacité d'écriture.
- Créez une politique de scalabilité automatique d'application pour `TestTable`. La politique vise à maintenir un ratio cible de 50 pour cent entre la capacité d'écriture consommée et la capacité d'écriture approvisionnée. La plage acceptée pour cette métrique est comprise entre 5 et 10 unités de capacité d'écriture. (La scalabilité automatique d'application n'est pas autorisée à ajuster le débit au-delà de cette plage.)
- Exécutez un programme Python pour générer du trafic d'écriture vers `TestTable`. Lorsque le ratio cible est supérieur à 50 pour cent pendant une période prolongée, la scalabilité automatique d'application demande à DynamoDB d'augmenter le débit de `TestTable` pour maintenir le taux d'utilisation cible de 50 %.
- Vérifiez que DynamoDB a correctement ajusté la capacité d'écriture approvisionnée pour `TestTable`.

> **Note**
>
> Vous pouvez également planifier la mise à l'échelle de DynamoDB de manière à ce qu'elle se produise à certains moments. Découvrez les étapes de base ici.

Rubriques

- Avant de commencer
- Étape 1 : création d'une table DynamoDB
- Étape 2 : enregistrer une cible évolutive
- Étape 3 : créer une politique de mise à l'échelle
- Étape 4 : Dirigez le trafic d'écriture vers TestTable
- Étape 5 : afficher les actions de scalabilité automatique d'application
- (Facultatif) Étape 6 : nettoyer

Avant de commencer

Avant de commencer à suivre le didacticiel, accomplissez les tâches suivantes :

Installer l'AWS CLI

Si vous ne l'avez pas déjà fait, vous devez installer et configurer AWS CLI. Pour ce faire, suivez les instructions du Guide de l'utilisateur AWS Command Line Interface :

- Installation de AWS CLI
- Configuration de l'interface AWS CLI (français non garanti)

Installation de Python

Une partie de ce didacticiel nécessite que vous exécutiez un programme Python (voir Étape 4 : Dirigez le trafic d'écriture vers TestTable). Si vous n'avez pas encore installé Python, vous pouvez télécharger Python..

Étape 1 : création d'une table DynamoDB

Au cours de cette étape, vous devez utiliser le AWS CLI pour créer`TestTable`. La clé primaire est composée de `pk` (clé de partition) et de `sk` (clé de tri). Ces deux attributs sont de type `Number`. Les paramètres de débit initial sont 5 unités de capacité de lecture et 5 unités de capacité d'écriture.

1. Utilisez la AWS CLI commande suivante pour créer la table.

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. Pour vérifier l'état de la table, utilisez la commande suivante.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   La table est prête pour utilisation quand son état est `ACTIVE`.

Étape 2 : enregistrer une cible évolutive

Ensuite, vous enregistrez la capacité d'écriture de la table en tant que cible évolutive pour la scalabilité automatique d'application. Cela permet à Application Auto Scaling d'ajuster la capacité d'écriture allouée pour TestTable, mais uniquement dans la plage de 5 à 10 unités de capacité.

> **Note**
>
> La scalabilité automatique de DynamoDB nécessite la présence d'un rôle lié à un service (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) qui effectue pour vous les actions de scalabilité automatique. Ce rôle est créé automatiquement pour vous. Pour plus d'informations, consultez Rôles liés aux services pour Application Auto Scaling dans le Guide de l'utilisateur Application Auto Scaling.

1. Entrez la commande suivante pour enregistrer la cible évolutive.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. Pour vérifier l'enregistrement, utilisez la commande suivante.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> Vous pouvez également enregistrer une cible évolutive par rapport à un index secondaire global. Par exemple, pour un index secondaire global (« test-index »), l'ID de ressource et les arguments de dimension évolutive sont mis à jour de manière appropriée.
>
> ```
> aws application-autoscaling register-scalable-target \
>     --service-namespace dynamodb \
>     --resource-id "table/TestTable/index/test-index" \
> ```

```
    --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

Étape 3 : créer une politique de mise à l'échelle

Dans cette étape, vous allez créer une politique de mise à l'échelle pour `TestTable`. La politique définit les détails des conditions dans lesquelles la scalabilité automatique d'application peut ajuster le débit approvisionné de votre table, et les actions à entreprendre à cet égard. Vous associez cette politique avec la cible évolutive que vous avez définie à l'étape précédente (unités de capacité d'écriture pour la table `TestTable`).

Cette politique contient les éléments suivants :

- `PredefinedMetricSpecification` – Métrique que la scalabilité automatique d'application est autorisée à ajuster. Pour DynamoDB, les valeurs suivantes sont valides pour `PredefinedMetricType` :
  - `DynamoDBReadCapacityUtilization`
  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown` – Durée minimum (en secondes) entre chaque événement de scalabilité automatique d'application qui augmente le débit approvisionné. Ce paramètre permet à la scalabilité automatique d'application d'augmenter de façon continue, mais pas agressive, le débit en réponse aux charges de travail réelles. Le paramètre par défaut pour `ScaleOutCooldown` est 0.
- `ScaleInCooldown` – Durée minimum (en secondes) entre chaque événement de scalabilité automatique d'application qui réduit le débit approvisionné. Ce paramètre permet à la scalabilité automatique d'application de réduire le débit progressivement et de façon prévisible. Le paramètre par défaut pour `ScaleInCooldown` est 0.
- `TargetValue` – La scalabilité automatique d'application garantit que le ratio de capacité consommée par rapport à la capacité approvisionnée reste égal ou proche de cette valeur. Vous définissez `TargetValue` en tant que pourcentage.

> **Note**
>
> Pour mieux comprendre le fonctionnement de `TargetValue`, supposons que vous ayez une table avec un paramètre de débit approvisionné de 200 unités de capacité d'écriture. Vous décidez de créer une politique de mise à l'échelle pour cette table, avec une `TargetValue` de 70 pour cent.
> Supposons maintenant que vous commencez à générer du trafic d'écriture vers la table de telle sorte que le débit d'écriture réel est de 150 unités de capacité. Le consumed-to-provisioned ratio est maintenant de (150/200), soit 75 %. Ce ratio dépassant votre cible, la scalabilité automatique d'application augmente la capacité d'écriture approvisionnée à 215, de façon que le ratio (150 / 215) soit de 69,77 %, aussi proche que possible de votre `TargetValue`, dans le dépasser.

Pour `TestTable`, vous définissez `TargetValue` sur 50 pour cent. Application Auto Scaling ajuste le débit provisionné de la table dans une fourchette de 5 à 10 unités de capacité (voirÉtape 2 : enregistrer une cible évolutive) afin que le consumed-to-provisioned ratio reste égal ou proche de 50 %. Vous définissez les valeurs pour `ScaleOutCooldown` et `ScaleInCooldown` sur 60 secondes.

1. Créez un fichier nommé `scaling-policy.json` avec les contenus suivants.

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Utilisez la AWS CLI commande suivante pour créer la politique.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
```

```
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. Dans le résultat, notez qu'Application Auto Scaling a créé deux CloudWatch alarmes Amazon, une pour chacune des limites supérieure et inférieure de la plage cible de dimensionnement.
4. Utilisez la AWS CLI commande suivante pour afficher plus de détails sur la politique de dimensionnement.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. Dans la sortie, vérifiez que les paramètres de politique correspondent à vos spécifications de Étape 2 : enregistrer une cible évolutive et Étape 3 : créer une politique de mise à l'échelle.

Étape 4 : Dirigez le trafic d'écriture vers TestTable

Vous pouvez maintenant tester votre politique de mise à l'échelle en écrivant des données dans `TestTable`. Pour ce faire, vous exécutez un programme Python.

1. Créez un fichier nommé `bulk-load-test-table.py` avec les contenus suivants.

```
import boto3
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
```

```
        j += 1
    i += 1
```

2. Pour exécuter le programme, entrez la commande suivante.

   ```
   python bulk-load-test-table.py
   ```

   La capacité d'écriture approvisionnée pour `TestTable` étant très faible (5 unités de capacité d'écriture), le programme se bloque occasionnellement en raison de la limitation d'écriture. Ce comportement est normal.

   Laissez le programme poursuivre son exécution pendant que vous passez à l'étape suivante.

Étape 5 : afficher les actions de scalabilité automatique d'application

Dans cette étape, vous allez afficher les actions de scalabilité automatique d'application qui ont été entreprises pour vous. Vous allez également vérifier que la scalabilité automatique d'application a mis à jour la capacité d'écriture approvisionnée pour `TestTable`.

1. Entrez la commande suivante pour afficher les actions de scalabilité automatique d'application.

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

   Réexécutez cette commande de temps en temps pendant l'exécution du programme Python (l'appel de votre politique de mise à l'échelle prend plusieurs minutes). Vous devriez voir la sortie suivante.

   ```
   ...
   {
       "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
       "Description": "Setting write capacity units to 10.",
       "ResourceId": "table/TestTable",
       "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
       "StartTime": 1489088210.175,
       "ServiceNamespace": "dynamodb",
       "EndTime": 1489088246.85,
       "Cause": "monitor alarm AutoScaling-table/TestTable-
   AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
    MyScalingPolicy",
   ```

```
    "StatusMessage": "Successfully set write capacity units to 10. Change
 successfully fulfilled by dynamodb.",
    "StatusCode": "Successful"
},
...
```

Cela indique que scalabilité automatique d'application a adressé une demande `UpdateTable` à DynamoDB.

2. Entrez la commande suivante pour vérifier que DynamoDB a augmenté la capacité d'écriture de la table.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

La `WriteCapacityUnits` aurait dû être augmentée de `5` à `10`.

(Facultatif) Étape 6 : nettoyer

Dans ce didacticiel, vous avez créé plusieurs ressources. Vous pouvez les supprimer si vous n'en avez plus besoin.

1. Supprimez la politique de mise à l'échelle pour `TestTable`.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Désenregistrez la cible évolutive

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Supprimez la table `TestTable`.

```
aws dynamodb delete-table --table-name TestTable
```

Utilisation du AWS SDK pour configurer le dimensionnement automatique sur les tables Amazon DynamoDB

En plus d'utiliser le AWS Management Console et le AWS Command Line Interface(AWS CLI), vous pouvez écrire des applications qui interagissent avec le dimensionnement automatique d'Amazon DynamoDB. Cette section deux programmes Java que vous pouvez utiliser pour tester cette fonctionnalité :

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Activation de la scalabilité automatique d’application pour une table

Le programme suivant présente un exemple de configuration d’une politique de scalabilité automatique pour une table DynamoDB (`TestTable`). Voici son mode d’action :

- Le programme enregistre les unités de capacité d’écriture en tant que cible évolutive pour `TestTable`. La plage acceptée pour cette métrique est comprise entre 5 et 10 unités de capacité d’écriture.
- Une fois la cible évolutive créée, le programme construit une configuration de suivi cible. La politique vise à maintenir un ratio cible de 50 pour cent entre la capacité d’écriture consommée et la capacité d’écriture approvisionnée.
- Le programme crée ensuite la politique de dimensionnement en fonction de la configuration de suivi cible.

> **Note**
>
> Lorsque vous supprimez manuellement une table ou une réplique de table globale, vous ne supprimez pas automatiquement les cibles évolutives, les politiques de dimensionnement ou les CloudWatch alarmes associées.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecification;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicyCon
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).
```

```
                roleARN - The ARN of the role that has ApplicationAutoScaling
permissions.
                policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table, which
is the resource to scale.");
        String tableId = args[0];
        String roleARN = args[1];
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
policyName);
    }

    public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
        try {
            RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .roleARN(roleARN)
                .minCapacity(5)
                .maxCapacity(10)
                .build();
```

```
            appAutoScalingClient.registerScalableTarget(targetRequest);
            System.out.println("You have registered " + tableId);

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the target was created.
    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    // Configure a scaling policy.
    public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
        // Check if the policy exists before creating a new one.
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
            System.out.println("Policy already exists. Consider updating it
instead.");
```

```
            List<ScalingPolicy> polList =
 describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
 PredefinedMetricSpecification.builder()

 .predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();

            TargetTrackingScalingPolicyConfiguration policyConfiguration =
 TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
 PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling policy
 for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " + e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

Java v1

Le programme requiert que vous fournissiez un Amazon Resource Name (ARN) pour un rôle lié à un service de scalabilité automatique d'application valide. (Par exemple : `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) Dans le programme suivant, remplacez `SERVICE_ROLE_ARN_GOES_HERE` par l'ARN réel.

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
```

```
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Define the scalable target
 RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withResourceId(resourceID)
   .withScalableDimension(tableWCUs)
   .withMinCapacity(5)
   .withMaxCapacity(10)
   .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

 try {
  aaClient.registerScalableTarget(rstRequest);
 } catch (Exception e) {
  System.err.println("Unable to register scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
```

```
    .withScaleInCooldown(60)
    .withScaleOutCooldown(60);

  // Create the scaling policy, based on your configuration
  PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy")
    .withPolicyType(PolicyType.TargetTrackingScaling)

 .withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

  try {
   aaClient.putScalingPolicy(pspRequest);
  } catch (Exception e) {
   System.err.println("Unable to put scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was created
  DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(dspRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

 }

}
```

Désactivation de la scalabilité automatique d'application pour une table

Le programme suivant inverse le processus précédent. Il supprime la politique Auto Scaling, puis désinscrit la cible évolutive.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s
```

```
            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
 ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
 appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
 ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
 DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");
```

```
        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();

        DescribeScalingPoliciesResponse response =
appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
```

```
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
```

```
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Delete the scaling policy
 DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy");

 try {
  aaClient.deleteScalingPolicy(delSPRequest);
 } catch (Exception e) {
  System.err.println("Unable to delete scaling policy: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scaling policy was deleted
 DescribeScalingPoliciesRequest descSPRequest = new
DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(descSPRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Remove the scalable target
 DeregisterScalableTargetRequest delSTRequest = new
DeregisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);
```

```
  try {
   aaClient.deregisterScalableTarget(delSTRequest);
  } catch (Exception e) {
   System.err.println("Unable to deregister scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scalable target was removed
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceIds(resourceID);

  try {
   DescribeScalableTargetsResult dsaResult =
 aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }

 }

}
```

Capacité réservée DynamoDB

Pour les tables à capacité provisionnée qui utilisent la classe de table Standard, DynamoDB offre la possibilité d'acheter de la capacité réservée pour votre capacité de lecture et d'écriture. Un achat de capacité réservée est un accord visant à payer un montant minimum de capacité de débit provisionnée, pendant la durée du contrat, en échange d'un prix réduit.

> **Note**
>
> Vous ne pouvez pas acheter de capacité réservée pour des unités de capacité d'écriture répliquées (rWCUs). La capacité réservée s'applique uniquement à la région dans laquelle

elle a été achetée. La capacité réservée n'est pas non plus disponible pour les tables utilisant la classe de tables DynamoDB Standard-IA ou le mode de capacité à la demande.

La capacité réservée est achetée par lots de 100 WCUs ou 100 RCUs. La plus petite offre de capacité réservée est de 100 unités de capacité (lecture ou écriture). La capacité réservée DynamoDB est proposée sous forme d'engagement d'un an ou, dans certaines régions, de trois ans. Vous pouvez économiser jusqu'à 54 % sur les tarifs standard pour une durée d'un an et 77 % pour une durée de trois ans. Pour plus d'informations sur la manière et le moment d'acheter cette capacité, consultez Capacité réservée Amazon DynamoDB.

Lorsque vous achetez de la capacité réservée DynamoDB, vous effectuez un paiement initial partiel unique et bénéficiez d'un tarif horaire réduit pour l'utilisation provisionnée à laquelle vous vous êtes engagé. Vous payez l'intégralité de l'utilisation provisionnée à laquelle vous vous êtes engagé, quelle que soit l'utilisation réelle, de sorte que vos économies sont étroitement liées à l'utilisation. Toute capacité que vous allouez au-delà de la capacité réservée achetée est facturée selon les frais de capacité provisionnée standard. En réservant à l'avance vos unités de capacité en lecture et à l'écriture, vous réalisez des économies importantes sur vos coûts de capacité allouée.

Vous ne pouvez pas vendre, annuler ou transférer de la capacité réservée vers une autre région ou un autre compte.

Présentation du débit chaud DynamoDB

Le débit chaud fait référence au nombre d'opérations de lecture et d'écriture que votre table DynamoDB peut prendre en charge instantanément. Ces valeurs sont disponibles par défaut pour toutes les tables et tous les index secondaires globaux (GSI) et indiquent dans quelle mesure ils ont été mis à l'échelle en fonction de leur utilisation historique. Si vous utilisez le mode à la demande ou si vous mettez à jour votre débit provisionné en fonction de ces valeurs, votre application sera en mesure d'émettre des demandes allant jusqu'à ces valeurs instantanément.

DynamoDB ajuste automatiquement les valeurs de débit chaud à mesure que votre utilisation augmente. Cependant, vous pouvez également augmenter ces valeurs de manière proactive en cas de besoin. Cela s'avère particulièrement utile lors des pics d'activité prévus, tels que les lancements de produits ou les ventes. Pour les pics planifiés, au cours desquels le taux de demandes adressées à votre table DynamoDB peut augmenter de 10, 100 fois ou plus, vous pouvez désormais évaluer si le débit chaud actuel est suffisant pour gérer le trafic attendu. Si ce n'est pas le cas, vous pouvez augmenter la valeur du débit chaud sans modifier vos paramètres de débit ni votre mode de

facturation. Ce processus est appelé préchauffage d'une table, ce qui vous permet de définir une base de référence que vos tables peuvent prendre en charge instantanément. Cela permet à vos applications de gérer des taux de demandes supérieurs dès leur apparition.

Vous pouvez augmenter la valeur du débit chaud pour les opérations de lecture, d'écriture ou les deux. Vous pouvez augmenter cette valeur pour les tables mono-régionales nouvelles et existantes, les tables globales et GSIs. Pour les tables globales, cette fonctionnalité est disponible pour la version 2019.11.21 (actuelle). Les paramètres de débit chaud que vous définissez s'appliquent automatiquement à toutes les tables répliquées de la table globale. Il n'existe pas de limite au nombre de tables DynamoDB que vous pouvez préchauffer à tout moment. Le temps nécessaire pour terminer le préchauffage dépend des valeurs que vous définissez et de la taille de la table ou de l'index. Vous pouvez envoyer des demandes de préchauffage simultanées, qui n'interféreront pas avec les opérations de table. Vous pouvez préchauffer votre table jusqu'à la limite du quota de table ou d'index de votre compte dans cette région. Utilisez la console Service Quotas pour vérifier vos limites actuelles et les augmenter si nécessaire.

Les valeurs de débit chaud sont disponibles gratuitement par défaut, pour toutes les tables et tous les index secondaires. Toutefois, si vous augmentez de manière proactive ces valeurs de débit chaud par défaut pour préchauffer les tables, ces demandes vous seront facturées. Pour plus d'informations, consultez Tarification Amazon DynamoDB.

Pour plus d'informations sur le débit chaud, consultez les rubriques ci-dessous :

Rubriques

- Vérification du débit chaud actuel de votre table DynamoDB
- Augmentation du débit chaud de votre table DynamoDB existante
- Création d'une table DynamoDB avec un débit chaud supérieur
- Compréhension du débit chaud DynamoDB dans différents scénarios

Vérification du débit chaud actuel de votre table DynamoDB

Utilisez les instructions suivantes AWS CLI et celles de AWS la console pour vérifier la valeur actuelle du débit à chaud de votre table ou de votre index.

AWS Management Console

Pour vérifier le débit chaud de votre table DynamoDB à l'aide de la console DynamoDB :

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Sur la page Tables, choisissez la table de votre choix.
4. Sélectionnez Paramètres supplémentaires pour afficher votre valeur de débit chaud actuelle. Cette valeur est affichée en unités de lecture et d'écriture par seconde.

AWS CLI

L'AWS CLI exemple suivant montre comment vérifier le débit chaud de votre table DynamoDB.

1. Exécutez l'opération `describe-table` sur votre table DynamoDB.

```
aws dynamodb describe-table --table-name GameScores
```

2. Vous recevrez une réponse similaire à la suivante. Vos paramètres `WarmThroughput` seront affichés sous la forme `ReadUnitsPerSecond` et `WriteUnitsPerSecond`. Le paramètre `Status` sera défini sur `UPDATING` lorsque la valeur du débit chaud sera mise à jour, et sur `ACTIVE` lorsque la nouvelle valeur du débit chaud sera définie.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
```

```
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1726128388.729,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST",
            "LastUpdateToPayPerRequestDateTime": 1726128388.729
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
```

```
                    "ReadCapacityUnits": 0,
                    "WriteCapacityUnits": 0
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 12000,
                    "WriteUnitsPerSecond": 4000,
                    "Status": "ACTIVE"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12000,
            "WriteUnitsPerSecond": 4000,
            "Status": "ACTIVE"
        }
    }
}
```

Augmentation du débit chaud de votre table DynamoDB existante

Une fois que vous avez vérifié la valeur actuelle du débit chaud de votre table DynamoDB, vous pouvez la mettre à jour en procédant comme suit :

AWS Management Console

Pour vérifier la valeur de débit chaud de votre table DynamoDB à l'aide de la console DynamoDB :

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation de gauche, choisissez Tables.
3. Sur la page Tables, choisissez la table de votre choix.
4. Dans le champ Débit chaud, sélectionnez Modifier.
5. Sur la page Modifier le débit chaud, choisissez Augmenter le débit chaud.
6. Ajustez les unités de lecture et d'écriture par seconde. Ces deux paramètres définissent le débit que votre table peut gérer instantanément.

7. Cliquez sur Enregistrer.
8. Vos unités de lecture par seconde et d'écriture par seconde seront mises à jour dans le champ Débit chaud lorsque le traitement de la demande sera terminé.

> **Note**
>
> La mise à jour de la valeur de votre débit chaud est une tâche asynchrone. La valeur de `Status` passera de `UPDATING` à `ACTIVE` lorsque la mise à jour sera terminée.

AWS CLI

L'AWS CLI exemple suivant montre comment mettre à jour la valeur de débit chaud de votre table DynamoDB.

1. Exécutez l'opération `update-table` sur votre table DynamoDB.

```
aws dynamodb update-table \
    --table-name GameScores \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --global-secondary-index-updates \
        "[
            {
                \"Update\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"WarmThroughput\": {
                        \"ReadUnitsPerSecond\": 88,
                        \"WriteUnitsPerSecond\": 77
                    }
                }
            }
        ]" \
    --region us-east-1
```

2. Vous recevrez une réponse similaire à la suivante. Vos paramètres `WarmThroughput` seront affichés sous la forme `ReadUnitsPerSecond` et `WriteUnitsPerSecond`. Le paramètre `Status` sera défini sur `UPDATING` lorsque la valeur du débit chaud sera mise à jour, et sur `ACTIVE` lorsque la nouvelle valeur du débit chaud sera définie.

```
{
    "TableDescription": {
```

```
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1730242189.965,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
```

```
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 50,
                    "WriteUnitsPerSecond": 25,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12300,
            "WriteUnitsPerSecond": 4500,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

Les exemples de SDK suivants vous montrent comment mettre à jour la valeur de débit chaud de votre table DynamoDB.

Java

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

...
public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}

public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long tableReadUnitsPerSecond,
                                       Long tableWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadUnitsPerSecond,
                                       Long globalSecondaryIndexWriteUnitsPerSecond)
 {

    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableReadUnitsPerSecond, tableWriteUnitsPerSecond);
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);

    final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                    .indexName(globalSecondaryIndexName)
                    .warmThroughput(gsiWarmThroughput)
                    .build()
            ).build();

    final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
```

```
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    System.out.println("Done!");
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def update_dynamodb_table_warm_throughput(table_name, table_read_units,
 table_write_units, gsi_name, gsi_read_units, gsi_write_units, region_name="us-
east-1"):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
```

```
            "WriteUnitsPerSecond": table_write_units
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {
                "Update": {
                    "IndexName": gsi_name,
                    "WarmThroughput": gsi_warm_throughput
                }
            }
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput
        }

        # Update the table
        ddb.update_table(**update_table_request)
        print("Table updated successfully!")
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
```

```
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${response}`);
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}
```

Création d’une table DynamoDB avec un débit chaud supérieur

Vous pouvez ajuster les valeurs de débit chaud lorsque vous créez votre table DynamoDB en suivant les étapes ci-dessous. Ces étapes s’appliquent également lors de la création d’une table globale ou d’un index secondaire.

AWS Management Console

Pour créer une table DynamoDB et ajuster les valeurs de débit chaud via la console :

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Sélectionnez Créer une table.
3. Choisissez le Nom de la table, une Clé de partition et une Clé de tri (facultatif).
4. Pour les Paramètres de la table, choisissez Personnaliser les paramètres.
5. Dans le champ Débit chaud, sélectionnez Augmenter le débit chaud.
6. Ajustez les unités de lecture et d'écriture par seconde. Ces deux paramètres définissent le débit maximal que votre table peut gérer instantanément.
7. Continuez à renseigner les autres détails de la table, puis sélectionnez Créer une table.

AWS CLI

L'AWS CLI exemple suivant montre comment créer une table DynamoDB avec des valeurs de débit de chaleur personnalisées.

1. Exécutez l'opération `create-table` pour créer la table DynamoDB suivante.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
```

```
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 50,
                    \"WriteCapacityUnits\": 25
                },\"WarmThroughput\": {
                    \"ReadUnitsPerSecond\": 1987,
                    \"WriteUnitsPerSecond\": 543
                }
            }
        ]" \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --region us-east-1
```

2. Vous recevrez une réponse similaire à la suivante. Vos paramètres `WarmThroughput` seront affichés sous la forme `ReadUnitsPerSecond` et `WriteUnitsPerSecond`. Le paramètre `Status` sera défini sur `UPDATING` lorsque la valeur du débit chaud sera mise à jour, et sur `ACTIVE` lorsque la nouvelle valeur du débit chaud sera définie.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1730241788.779,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 1987,
```

```
                    "WriteUnitsPerSecond": 543,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12345,
            "WriteUnitsPerSecond": 4567,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

L'exemple de kit SDK suivant montre comment créer une table DynamoDB avec des valeurs de débit chaud personnalisées.

Java

```
import software.amazon.awscdk.services.dynamodb.ProjectionType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
...

public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}
```

```
private static AttributeDefinition buildAttributeDefinition(final String
 attributeName,
                                                             final
 ScalarAttributeType scalarAttributeType) {
    return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
}
private static KeySchemaElement buildKeySchemaElement(final String attributeName,
                                                      final KeyType keyType) {
    return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
}
public static void createDynamoDBTable(DynamoDbClient ddb,
                                        String tableName,
                                        String partitionKey,
                                        String sortKey,
                                        String miscellaneousKeyAttribute,
                                        String nonKeyAttribute,
                                        Long tableReadCapacityUnits,
                                        Long tableWriteCapacityUnits,
                                        Long tableWarmReadUnitsPerSecond,
                                        Long tableWarmWriteUnitsPerSecond,
                                        String globalSecondaryIndexName,
                                        Long globalSecondaryIndexReadCapacityUnits,
                                        Long globalSecondaryIndexWriteCapacityUnits,
                                        Long
 globalSecondaryIndexWarmReadUnitsPerSecond,
                                        Long
 globalSecondaryIndexWarmWriteUnitsPerSecond) {

    // Define the table attributes
    final AttributeDefinition partitionKeyAttribute =
 buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
    final AttributeDefinition sortKeyAttribute = buildAttributeDefinition(sortKey,
 ScalarAttributeType.S);
    final AttributeDefinition miscellaneousKeyAttributeDefinition =
 buildAttributeDefinition(miscellaneousKeyAttribute, ScalarAttributeType.N);
    final AttributeDefinition[] attributeDefinitions = {partitionKeyAttribute,
 sortKeyAttribute, miscellaneousKeyAttributeDefinition};
```

```
   // Define the table key schema
   final KeySchemaElement partitionKeyElement = buildKeySchemaElement(partitionKey,
KeyType.HASH);
   final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
   final KeySchemaElement[] keySchema = {partitionKeyElement, sortKeyElement};

   // Define the provisioned throughput for the table
   final ProvisionedThroughput provisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(tableReadCapacityUnits)
           .writeCapacityUnits(tableWriteCapacityUnits)
           .build();

   // Define the Global Secondary Index (GSI)
   final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
   final KeySchemaElement globalSecondaryIndexSortKeyElement =
buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
   final KeySchemaElement[] gsiKeySchema =
{globalSecondaryIndexPartitionKeyElement, globalSecondaryIndexSortKeyElement};

   final Projection gsiProjection = Projection.builder()
           .projectionType(String.valueOf(ProjectionType.INCLUDE))
           .nonKeyAttributes(nonKeyAttribute)
           .build();
   final ProvisionedThroughput gsiProvisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(globalSecondaryIndexReadCapacityUnits)
           .writeCapacityUnits(globalSecondaryIndexWriteCapacityUnits)
           .build();
   // Define the warm throughput for the Global Secondary Index (GSI)
   final WarmThroughput gsiWarmThroughput =
buildWarmThroughput(globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);
   final GlobalSecondaryIndex globalSecondaryIndex = GlobalSecondaryIndex.builder()
           .indexName(globalSecondaryIndexName)
           .keySchema(gsiKeySchema)
           .projection(gsiProjection)
           .provisionedThroughput(gsiProvisionedThroughput)
           .warmThroughput(gsiWarmThroughput)
           .build();

   // Define the warm throughput for the table
```

```
    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableWarmReadUnitsPerSecond, tableWarmWriteUnitsPerSecond);

    final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

    CreateTableResponse response = ddb.createTable(request);
    System.out.println(response);
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def create_dynamodb_table_warm_throughput(table_name, partition_key,
 sort_key, misc_key_attr, non_key_attr, table_provisioned_read_units,
 table_provisioned_write_units, table_warm_reads, table_warm_writes, gsi_name,
 gsi_provisioned_read_units, gsi_provisioned_write_units, gsi_warm_reads,
 gsi_warm_writes, region_name="us-east-1"):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned read
 capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's warm
 throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
```

```
   :param gsi_name: The name of the Global Secondary Index (GSI) to be created on
the table.
   :param gsi_provisioned_read_units: The configured Global Secondary Index (GSI)
provisioned read capacity units.
   :param gsi_provisioned_write_units: The configured Global Secondary Index (GSI)
provisioned write capacity units.
   :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param region_name: The AWS Region name to target. defaults to us-east-1
   """
   try:
       ddb = resource('dynamodb', region_name)

       # Define the table attributes
       attribute_definitions = [
           { "AttributeName": partition_key, "AttributeType": "S" },
           { "AttributeName": sort_key, "AttributeType": "S" },
           { "AttributeName": misc_key_attr, "AttributeType": "N" }
       ]

       # Define the table key schema
       key_schema = [
           { "AttributeName": partition_key, "KeyType": "HASH" },
           { "AttributeName": sort_key, "KeyType": "RANGE" }
       ]

       # Define the provisioned throughput for the table
       provisioned_throughput = {
           "ReadCapacityUnits": table_provisioned_read_units,
           "WriteCapacityUnits": table_provisioned_write_units
       }

       # Define the global secondary index
       gsi_key_schema = [
           { "AttributeName": sort_key, "KeyType": "HASH" },
           { "AttributeName": misc_key_attr, "KeyType": "RANGE" }
       ]
       gsi_projection = {
           "ProjectionType": "INCLUDE",
           "NonKeyAttributes": [non_key_attr]
       }
       gsi_provisioned_throughput = {
```

```
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput
        )

        print(response)
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

async function createDynamoDBTableWithWarmThroughput(
```

```
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
```

```
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}
```

Compréhension du débit chaud DynamoDB dans différents scénarios

Voici quelques scénarios que vous êtes susceptible de rencontrer lors de l'utilisation du débit chaud DynamoDB.

Rubriques

- Débit chaud et modèles d'accès inégaux
- Débit chaud pour une table provisionnée
- Débit chaud pour une table à la demande
- Débit chaud pour une table à la demande avec un débit maximal configuré

Débit chaud et modèles d'accès inégaux

Une table peut avoir un débit chaud de 30 000 unités de lecture par seconde et de 10 000 unités d'écriture par seconde, mais vous pouvez tout de même ressentir une limitation des lectures ou des écritures avant d'atteindre ces valeurs. Cela est probablement dû à une partition chaude. Bien que

DynamoDB puisse continuer à évoluer pour prendre en charge un débit pratiquement illimité, chaque partition individuelle est limitée à 1 000 unités d'écriture par seconde et à 3 000 unités de lecture par seconde. Si votre application génère trop de trafic vers une petite partie des partitions de la table, la limitation peut avoir lieu avant même que vous n'atteigniez les valeurs de débit chaud de la table. Nous vous recommandons de suivre les Bonnes pratiques DynamoDB afin de garantir une capacité de mise à l'échelle sans faille et d'éviter les partitions critiques.

Débit chaud pour une table provisionnée

Imaginons une table provisionnée dont le débit chaud est de 30 000 unités de lecture par seconde et de 10 000 unités d'écriture par seconde, mais dont le débit provisionné est actuellement de 4 000 RCU et 8 000 WCU. Vous pouvez mettre à l'échelle instantanément le débit provisionné de la table jusqu'à 30 000 RCU ou 10 000 WCU en mettant à jour vos paramètres de débit provisionné. Lorsque vous augmentez le débit provisionné au-delà de ces valeurs, le débit chaud s'ajuste automatiquement aux nouvelles valeurs supérieures, car vous avez établi un nouveau débit de pointe. Par exemple, si vous définissez le débit provisionné sur 50 000 RCU, le débit chaud augmentera pour atteindre 50 000 unités de lecture par seconde.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Débit chaud pour une table à la demande

Une nouvelle table à la demande utilise au début un débit chaud de 12 000 unités de lecture par seconde et de 4 000 unités d'écriture par seconde. Votre table peut instantanément accueillir un trafic soutenu jusqu'à ces niveaux. Lorsque vos demandes dépassent 12 000 unités de lecture par seconde ou 4 000 unités d'écriture par seconde, le débit chaud s'ajuste automatiquement aux valeurs les plus élevées.

```
"WarmThroughput":
    {
```

```
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Débit chaud pour une table à la demande avec un débit maximal configuré

Imaginons une table à la demande avec un débit moyen de 30 000 unités de lecture par seconde, mais avec un débit maximal configuré à 5 000 unités de demande de lecture (RRU). Dans ce scénario, le débit de la table sera limité au maximum de 5000 RRU que vous avez défini. Toute demande de débit dépassant ce maximum sera limitée. Cependant, vous pouvez modifier le débit maximal spécifique à la table à tout moment, en fonction des besoins de votre application.

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Débordement et capacité adaptative DynamoDB

Pour minimiser la limitation liée aux exceptions de débit, DynamoDB utilise la capacité de débordement pour gérer les pics d'utilisation. DynamoDB utilise la capacité adaptative pour s'adapter aux modèles d'accès inégaux.

Capacité de débordement

DynamoDB vous offre une certaine souplesse dans l'allocation du débit en fournissant une capacité de débordement. Chaque fois que vous n'utilisez pas totalement le débit disponible, DynamoDB réserve une partie de cette capacité inutilisée pour des débordements de débit ultérieurs afin de pouvoir gérer les pics d'utilisation. La capacité de transmission en mode rafale permet de satisfaire les demandes inattendues en lecture ou en écriture, alors qu'elles seraient sinon limitées.

DynamoDB conserve actuellement jusqu'à cinq minutes (300 secondes) de capacité en lecture et en écriture non utilisée. En cas de débordement occasionnel d'activités de lecture ou d'écriture, ces

unités de capacité supplémentaires peuvent être consommées rapidement, et même plus rapidement que la capacité de débit approvisionnée par seconde que vous avez définie pour votre table.

DynamoDB peut également utiliser la capacité de débordement pour la maintenance en arrière-plan et d'autres tâches effectuées sans préavis.

Notez que ces informations relatives à la capacité de transmission en mode rafale sont susceptibles d'évoluer.

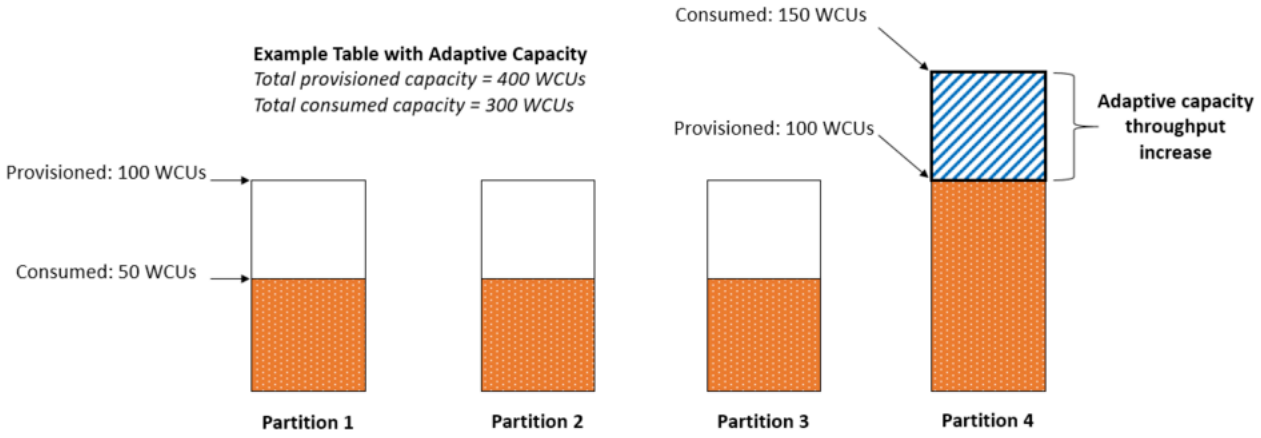
Capacité adaptative

DynamoDB répartit automatiquement vos données entre des partitions, qui sont stockées sur plusieurs serveurs dans le AWS Cloud. Il n'est pas toujours possible de répartir les activités de lecture et d'écriture uniformément. Lorsque l'accès aux données est déséquilibré, une partition « en surchauffe » peut recevoir un volume de lectures et d'écritures bien supérieur à celui des autres partitions. Comme les opérations de lecture et d'écriture sur une partition sont gérées indépendamment, la limitation se produit si une seule partition reçoit plus de 3 000 opérations de lecture ou plus de 1 000 opérations d'écriture. La capacité adaptative fonctionne en augmentant automatiquement la capacité de débit des partitions qui reçoivent le plus de trafic.

Pour mieux s'adapter aux modèles d'accès irréguliers, la capacité adaptative de DynamoDB permet à votre application de poursuivre les opérations de lecture et d'écriture sur des partitions critiques sans limitation, tant que le trafic ne dépasse pas la capacité totale approvisionnée de votre table ou la capacité maximale de la partition. La capacité adaptative fonctionne en augmentant automatiquement et instantanément la capacité de débit des partitions qui reçoivent le plus de trafic.

Le graphique suivant illustre le fonctionnement de la capacité adaptative. La table d'exemple est provisionnée avec 400 unités WCUs réparties uniformément sur quatre partitions, ce qui permet à chaque partition d'en supporter jusqu'à 100 WCUs par seconde. Les partitions 1, 2 et 3 reçoivent chacune un trafic d'écriture de 50WCU/sec. Partition 4 receives 150 WCU/sec. This hot partition can accept write traffic while it still has unused burst capacity, but eventually it throttles traffic that exceeds 100 WCU/sec.

La capacité adaptative de DynamoDB répond en augmentant la capacité de la partition 4 afin qu'elle puisse supporter la charge de travail plus élevée de WCU/sec 150 sans être limitée.



La capacité adaptative est activée automatiquement pour chaque table DynamoDB sans coût supplémentaire. Vous n'avez pas besoin de l'activer ou de la désactiver explicitement.

Isoler les éléments fréquemment consultés

Si votre application génère un trafic excessivement élevé vers un ou plusieurs éléments, la capacité adaptative rééquilibre vos partitions de manière à ce que les éléments fréquemment utilisés ne résident pas sur la même partition. L'isolation des éléments fréquemment consultés réduit les possibilités de limitation des demandes en raison d'une charge de travail qui dépasse le quota de débit sur une seule partition. Vous pouvez également diviser une collection d'éléments en segments par clé de tri, tant que la collection d'articles ne représente pas un trafic suivi par une augmentation ou une diminution monotone de la clé de tri.

Si votre application dirige systématiquement le trafic élevé vers un seul élément, la capacité adaptative peut rééquilibrer vos données de façon à ce qu'une partition contienne seulement cet élément unique fréquemment consulté. Dans ce cas, DynamoDB peut fournir un débit maximal de partition de RCUs 3 000 et WCUs 1 000 à la clé primaire de cet élément. La capacité adaptative ne fractionne pas les collections d'éléments entre plusieurs partitions de la table lorsqu'il existe un index secondaire localsur la table.

Considérations relatives au changement de mode de capacité dans DynamoDB

Lorsque vous créez une table DynamoDB, vous devez sélectionner le mode de capacité à la demande ou de capacité provisionnée.

Vous pouvez faire passer les tables du mode de capacité provisionnée au mode à la demande jusqu'à quatre fois par période de 24 heures. Vous pouvez à tout moment faire passer des tables du mode à la demande au mode de capacité provisionnée.

Rubriques

- Passage du mode de capacité provisionné à celui à la demande
- Passage du mode de capacité à la demande à celui provisionné

Passage du mode de capacité provisionné à celui à la demande

En mode provisionné, vous définissez la capacité de lecture et d'écriture en fonction des besoins attendus de votre application. Lorsque vous passez du mode approvisionné au mode à la demande pour une table, vous devez préciser le débit de lecture et d'écriture que votre application devrait atteindre. DynamoDB à la demande propose une tarification pay-per-request simple pour les demandes de lecture et d'écriture, de sorte que vous ne payez que pour ce que vous utilisez, ce qui facilite l'équilibre entre les coûts et les performances. Vous pouvez éventuellement configurer le débit maximal de lecture ou d'écriture (ou les deux) pour les tables individuelles à la demande et les index secondaires globaux associés, afin de vous aider à maîtriser les coûts et l'utilisation. Pour plus d'informations sur la définition du débit maximal pour une table ou un index spécifique, consultez Débit maximal DynamoDB pour les tables à la demande.

Lorsque vous changez le mode pour passer d'une capacité allouée à une capacité à la demande, DynamoDB apporte plusieurs changements à la structure de la table et des partitions. Ce processus peut prendre plusieurs minutes. Pendant la période de basculement, votre table fournit un débit correspondant aux volumes d'unités de capacité en écriture et en lecture alloués antérieurement.

Débit initial pour le mode de capacité à la demande

Si vous avez récemment activé le mode de capacité à la demande sur une table existante, la table possède les paramètres suivants de trafic de pointe précédent, même si la table n'a pas encore opéré de trafic en mode de capacité à la demande.

Vous trouverez ci-dessous des exemples de scénarios possibles :

- Toute table provisionnée configurée en dessous de 4 000 WCU et 12 000 RCU, qui n'a jamais été provisionnée pour une valeur supérieure. Lorsque vous passez cette table à la demande pour la première fois, DynamoDB veille à ce qu'elle soit redimensionnée pour prendre en charge instantanément au moins 4 000 unités d' units/sec écriture et 12 000 unités de lecture par seconde.

- Une table provisionnée configurée en 8 000 WCU et 24 000 RCU. Lorsque vous passez cette table en mode à la demande, elle pourra continuer à supporter au moins 8 000 écritures units/sec et 24 000 lectures units/sec à tout moment.
- Une table provisionnée configurée avec 8 000 WCU et 24 000 RCU, qui a consommé 6 000 écritures units/sec et 18 000 lectures units/sec pendant une période prolongée. Lorsque vous passez cette table en mode à la demande, elle continuera à supporter au moins 8 000 unités d'écriture units/sec et 24 000 unités de lecture par seconde. Le trafic précédent peut également permettre à la table de maintenir des niveaux de trafic beaucoup plus élevés sans limitation.
- Table précédemment provisionnée avec 10 000 WCU et 10 000 RCU, mais actuellement provisionnée avec 10 RCU et 10 WCU. Lorsque vous passez cette table en mode à la demande, elle pourra supporter au moins 10 000 unités d'écriture units/sec et 10 000 unités de lecture par seconde.

Paramètres d’autoscaling

Lorsque vous faites passer une table du mode approvisionné au mode à la demande :

- Si vous utilisez la console, tous vos paramètres de scalabilité automatique (éventuels) seront supprimés.
- Si vous utilisez le AWS SDK AWS CLI ou le SDK, tous vos paramètres de mise à l'échelle automatique seront conservés. Ils peuvent être appliqués lorsque vous mettez à jour la table pour la faire repasser en mode de facturation approvisionné.

Modification groupée du mode de capacité dans la console DynamoDB

Vous pouvez modifier plusieurs tables de manière groupée pour passer du mode capacité provisionné au mode de capacité à la demande à l’aide de la console DynamoDB. Pour modifier le mode de capacité de manière groupée :

1. Ouvrez la page Tables dans la console DynamoDB.
2. Cochez les cases des tables que vous souhaitez mettre à jour vers le mode de capacité à la demande.
3. Choisissez Actions, puis sélectionnez Mettre à jour vers le mode de capacité à la demande.

Cette opération groupée vous permet de basculer efficacement plusieurs tables en mode capacité à la demande sans avoir à mettre à jour chaque table individuellement.

Passage du mode de capacité à la demande à celui provisionné

Si vous passez du mode de capacité à la demande vers le mode de capacité allouée, votre table fournit un débit correspondant au trafic de pointe précédent atteint lorsque la table était en mode de capacité à la demande.

Gestion des capacités

Prenez en compte les considérations suivantes lorsque vous faites passer une table du mode à la demande au mode approvisionné :

- Si vous utilisez le AWS SDK AWS CLI ou le SDK, choisissez les bons paramètres de capacité provisionnée pour votre table et vos index secondaires globaux en utilisant Amazon CloudWatch pour examiner votre consommation historique (`ConsumedWriteCapacityUnits`et vos `ConsumedReadCapacityUnits` indicateurs) afin de déterminer les nouveaux paramètres de débit.

  > **Note**
  >
  > Si vous faites passer une table globale au mode approvisionné, vérifiez votre consommation maximale sur tous vos réplicas pour les tables de base et les index secondaires globaux afin de déterminer les nouveaux paramètres de débit.

- Si vous repassez du mode à la demande au mode provisionné, veillez à définir les unités initialement provisionnées à un niveau suffisamment élevé pour gérer votre capacité de table ou d'index pendant la transition.

Gestion de la scalabilité automatique

Lorsque vous mettez à jour une table pour la faire repasser du mode à la demande au mode approvisionné :

- Si vous utilisez la console, nous vous recommandons d'activer l'autoscaling avec les valeurs par défaut suivantes :
  - Utilisation cible : 70%
  - Capacité allouée minimum : 5 unités
  - Capacité allouée maximum : le maximum de la région

- Si vous utilisez le SDK AWS CLI ou le SDK, vos précédents paramètres de mise à l'échelle automatique (le cas échéant) sont conservés.

Programmation avec DynamoDB et AWS SDKs

Cette section contient des rubriques destinées aux développeurs. Si vous voulez plutôt exécuter des exemples de code, consultez Exécution des exemples de code du Guide du développeur.

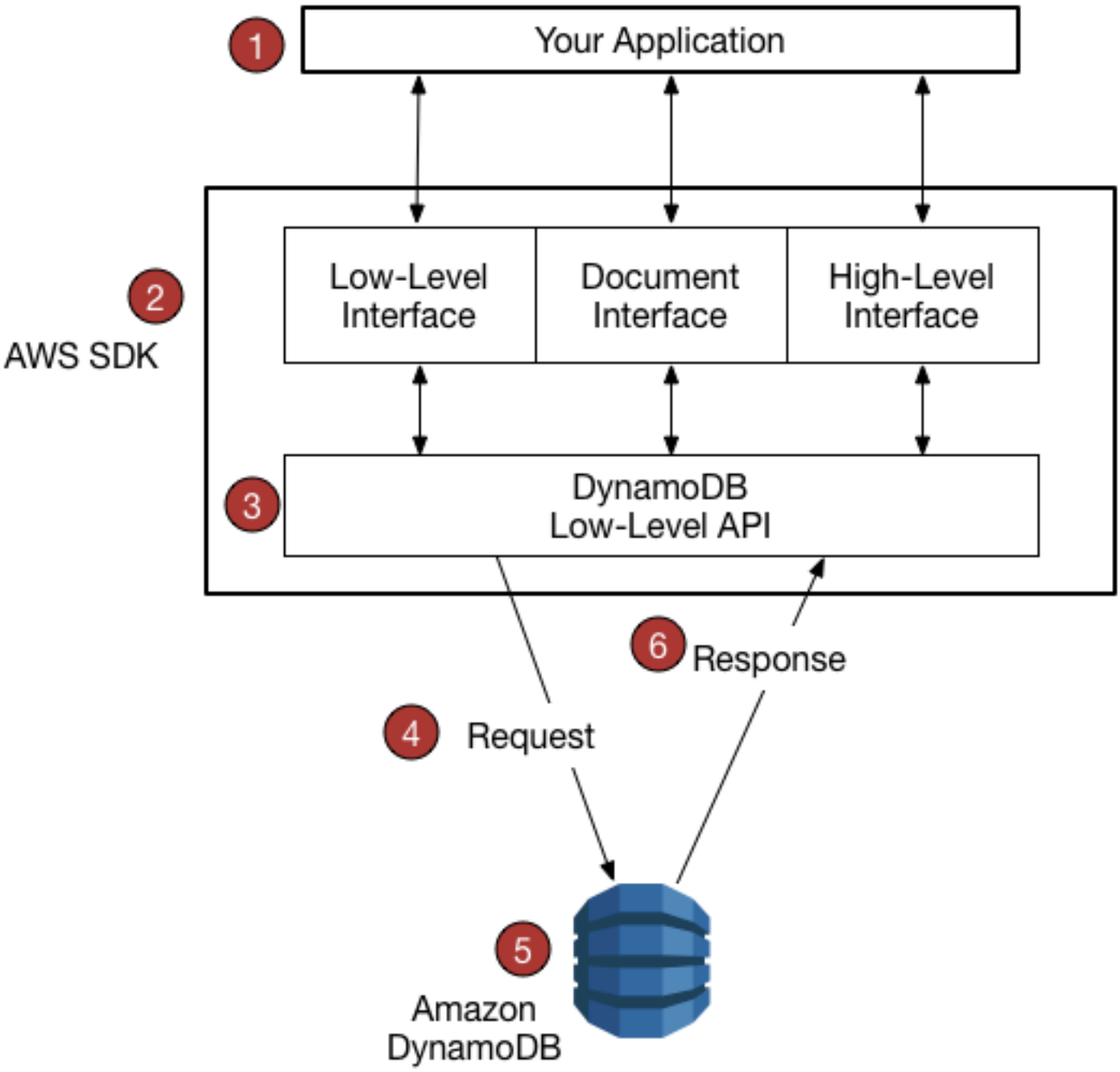
> **Note**
>
> En décembre 2017, le processus de migration de tous les points de terminaison Amazon DynamoDB AWS a débuté afin d'utiliser des certificats sécurisés émis par Amazon Trust Services (ATS). Pour de plus amples informations, veuillez consulter Résolution des problèmes d'établissement de SSL/TLS connexion avec DynamoDB.

Rubriques

- Présentation de la prise en charge du AWS SDK pour DynamoDB
- Programmation d'Amazon DynamoDB avec Python et Boto3
- Programmation d'Amazon DynamoDB avec JavaScript
- Programmation de DynamoDB avec AWS SDK for Java 2.x
- Gestion des erreurs avec DynamoDB
- Utilisation de DynamoDB avec un SDK AWS

Présentation de la prise en charge du AWS SDK pour DynamoDB

Le schéma suivant fournit une présentation générale de la programmation d'applications Amazon DynamoDB à l'aide du.AWS SDKs



1. Vous écrivez une application à l'aide d'un AWS SDK pour votre langage de programmation.
2. Chaque AWS SDK fournit une ou plusieurs interfaces de programmation permettant d'utiliser DynamoDB. Les interfaces spécifiques disponibles dépendent du langage de programmation et du AWS SDK que vous utilisez. Voici les options :
   - Interfaces de bas niveau compatibles avec DynamoDB
   - Interfaces de document compatibles avec DynamoDB
   - Interfaces de persistance d'objets compatibles avec DynamoDB

- Interfaces de haut niveau

3. Le AWS SDK crée des requêtes HTTP (S) à utiliser avec l'API DynamoDB de bas niveau.
4. Le AWS SDK envoie la demande au point de terminaison DynamoDB.
5. DynamoDB exécute la demande. Si la demande aboutit, DynamoDB renvoie un code de réponse HTTP 200 (OK). Si la demande échoue, DynamoDB renvoie un code d'erreur HTTP et un message d'erreur.
6. Le AWS SDK traite la réponse et la retransmet à votre application.

Chacun d'entre eux AWS SDKs fournit des services importants à votre application, notamment les suivants :

- Mise en forme des demandes HTTP(S) et sérialisation des paramètres de demande.
- Génération d'une signature de chiffrement pour chaque demande.
- Transfert des demandes vers un point de terminaison DynamoDB et réception des réponses de DynamoDB.
- Extraction des résultats de ces réponses.
- Implémentation d'une logique de nouvelle tentative de base en cas d'erreurs.

Vous n'avez besoin d'écrire de code pour aucune de ces tâches.

> **Note**
>
> Pour plus d'informations AWS SDKs, y compris les instructions d'installation et la documentation, consultez la section Outils pour Amazon Web Services.

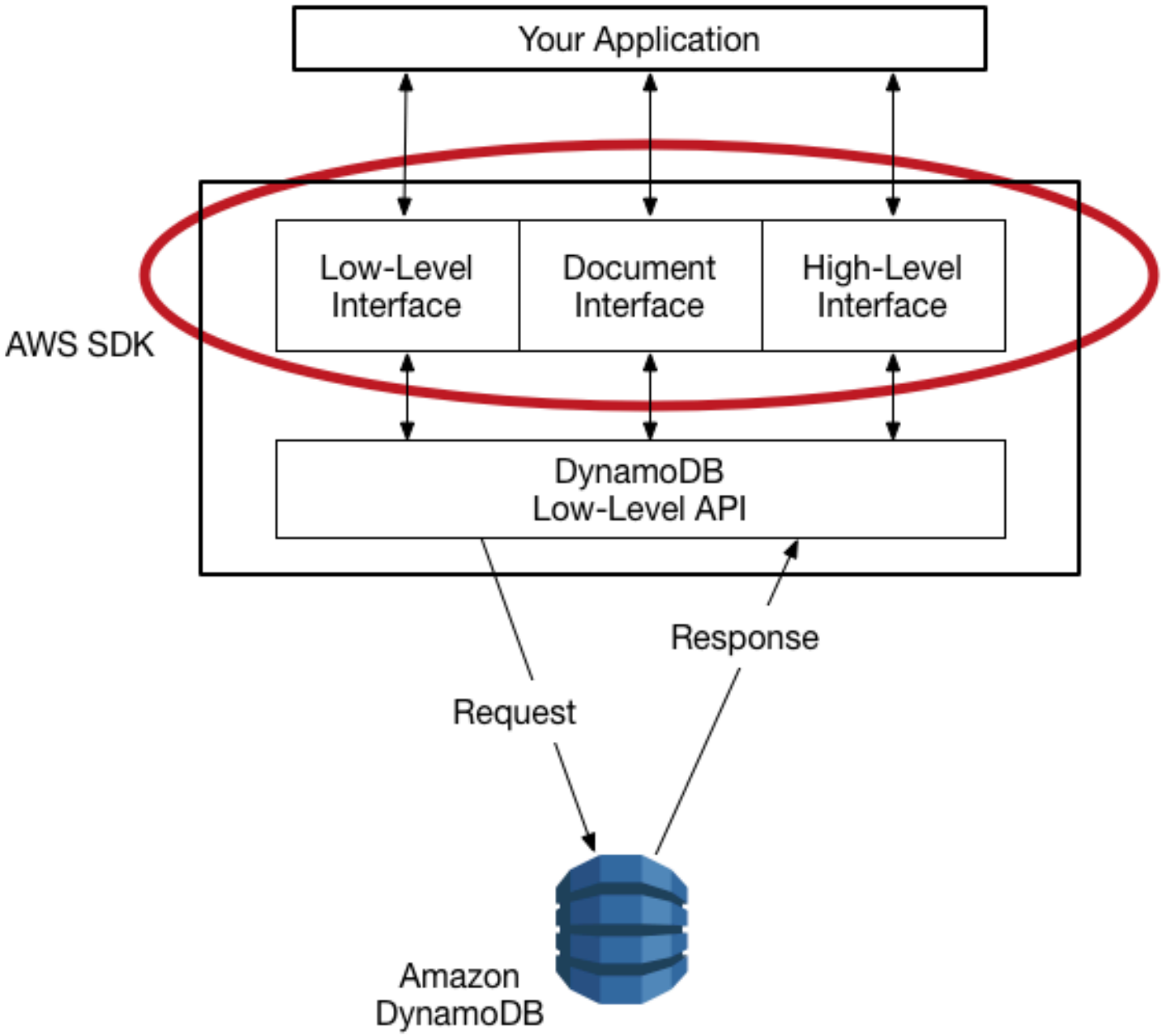
Support du SDK pour les points de terminaison basés sur des AWS comptes

AWS déploie la prise en charge du SDK pour les AWS points de terminaison basés sur un compte pour DynamoDB, en commençant par le SDK pour AWS Java V1 le 4 septembre 2024. Ces nouveaux terminaux contribuent AWSà garantir des performances et une évolutivité élevées. La mise à jour SDKs utilisera automatiquement les nouveaux points de terminaison, qui ont le format`https://(account-id).ddb.(region).amazonaws.com`.

Si vous utilisez une seule instance d'un client SDK pour envoyer des demandes à plusieurs comptes, votre application aura moins de possibilités de réutiliser les connexions.AWS recommande de modifier vos applications afin de vous connecter à un moins grand nombre de comptes par instance client SDK. Une autre solution consiste à configurer votre client SDK pour qu'il continue à utiliser les points de terminaison régionaux à l'aide du `ACCOUNT_ID_ENDPOINT_MODE` paramètre, comme indiqué dans le guide de référence AWS SDKs et des outils.

Interfaces de programmation compatibles avec DynamoDB

Chaque kit SDK AWS fournit une ou plusieurs interfaces de programmation utilisables avec Amazon DynamoDB. Ces interfaces vont de simples encapsuleurs DynamoDB de bas niveau à des couches de persistance orientées objet. Les interfaces disponibles varient en fonction du AWS SDK et du langage de programmation que vous utilisez.



La section suivante met en évidence certaines des interfaces disponibles, en utilisant l'AWS SDK pour Java en guise d'exemple (Toutes les interfaces ne sont pas toutes disponibles AWS SDKs.)

Rubriques

- Interfaces de bas niveau compatibles avec DynamoDB
- Interfaces de document compatibles avec DynamoDB
- Interfaces de persistance d'objets compatibles avec DynamoDB

Interfaces de bas niveau compatibles avec DynamoDB

Chaque AWS SDK spécifique à un langage fournit une interface de bas niveau pour Amazon DynamoDB, avec des méthodes qui ressemblent étroitement aux requêtes d'API DynamoDB de bas niveau.

Dans certains cas, vous devrez identifier les types de données des attributs en utilisant des Descripteurs de type de données, tels que `S` pour String (chaîne) ou `N` pour Number (nombre).

> **Note**
>
> Une interface de bas niveau est disponible dans chaque kit SDK AWS spécifique d'un langage.

Le programme Java suivant utilise l'interface de bas niveau de l'AWS SDK pour Java.

Exemple d'interface de bas niveau

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();
```

```
        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Interfaces de document compatibles avec DynamoDB

Beaucoup AWS SDKs proposent une interface documentaire qui vous permet d'effectuer des opérations de plan de données (création, lecture, mise à jour, suppression) sur les tables et les index. Avec une interface de document, vous n'avez pas besoin de spécifier Descripteurs de type de données. Les types de données découlent de la sémantique des données proprement dites. Ils fournissent AWS SDKs également des méthodes permettant de convertir facilement des documents JSON vers et depuis des types de données Amazon DynamoDB natifs.

> **Note**
>
> Les interfaces de document sont disponibles dans Java, .NET, Node.js et JavaScriptSDK.AWS SDKs

Le programme Java suivant utilise l'interface de document de l'AWS SDK pour Java. Le programme crée un objet `Table` qui représente la table `Music`, puis demande à cet objet d'utiliser `GetItem` pour extraire une chanson. Le programme affiche ensuite l'année de sortie de la chanson.

La classe `software.amazon.dynamodb.document.DynamoDB` implémente l'interface de document de DynamoDB. Observez la manière dont `DynamoDB` agit en tant qu'encapsuleur autour du client de bas niveau (`AmazonDynamoDB`).

Exemple d'interface de document

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

Interfaces de persistance d'objets compatibles avec DynamoDB

Certains AWS SDKs proposent une interface de persistance des objets dans laquelle vous n'effectuez pas directement d'opérations sur le plan de données. Au lieu de cela, vous créez des objets représentant des éléments dans des tables et index Amazon DynamoDB, et interagissez uniquement avec ces objets. Cela vous permet d'écrire du code orienté objet plutôt que du code orienté base de données.

> **Note**
>
> Les interfaces de persistance des objets sont disponibles AWS SDKs pour Java et .NET. Pour plus d'informations, consultez Interfaces de programmation de niveau supérieur pour DynamoDB pour DynamoDB.

Exemple d'interface de persistance des objets

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
```

```
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
```

```
    }
}
```

Interfaces de programmation de niveau supérieur pour DynamoDB

Les kits SDK AWS fournissent des applications dotées d'interfaces de bas niveau pour utiliser Amazon DynamoDB. Ces classes et méthodes côté client correspondent directement à l'API DynamoDB de bas niveau. Cependant, de nombreux développeurs sont confrontés à un sentiment de déconnexion ou à une discordance d'impédance quand ils doivent mapper des types de données complexes à des éléments dans une table de base de données. Avec une interface de base de données de bas niveau, les développeurs doivent écrire des méthodes pour lire ou écrire des données d'objet dans des tables de base de données, et inversement. La quantité de code supplémentaire requise pour chaque combinaison de type d'objet et de table de base de données peut sembler écrasante.

Pour simplifier le développement, les kits SDK AWS pour Java et .NET fournissent des interfaces supplémentaires avec des niveaux d'abstraction plus élevés. Les interfaces de niveau supérieur pour DynamoDB vous permettent de définir les relations entre les objets dans votre programme et les tables de base de données qui stockent les données de ces objets. Après avoir défini ce mappage, vous appelez des méthodes d'objet simples telles que `save`, `load` ou `delete`, et les opérations DynamoDB de bas niveau sous-jacentes sont appelées automatiquement pour vous. Cela vous permet d'écrire du code orienté objet plutôt que du code orienté base de données.

Les interfaces de programmation de niveau supérieur pour DynamoDB sont disponibles dans les kits SDK AWS pour Java et .NET.

Java

- Java 1.x : DynamoDBMapper
- Java 2.x : client amélioré DynamoDB

.NET

- Utilisation du modèle de document .NET dans DynamoDB
- Utilisation du modèle de persistance des objets .NET et de DynamoDB

Java 1.x : DynamoDBMapper

> **Note**
>
> Le kit SDK pour Java est disponible en deux versions : 1.x et 2.x. La fin du support de la version 1.x a été annoncée le 12 janvier 2024. Cela est confirmé et la fin de support est prévue pour le 31 décembre 2025. Pour les nouveaux développements, nous vous recommandons vivement d'utiliser la version 2.x.

L'AWS SDK pour Java fournit une classe `DynamoDBMapper` qui vous permet de mapper vos classes côté client à des tables Amazon DynamoDB. Pour utiliser `DynamoDBMapper`, vous définissez la relation entre éléments d'une table DynamoDB et leurs instances d'objet correspondantes dans votre code. La classe `DynamoDBMapper` vous permet d'effectuer diverses opérations de création, de lecture, de mise à jour et de suppression (opérations CRUD) sur des éléments, ainsi que d'exécuter des requêtes et analyses sur des tables.

Rubriques

- Classe DynamoDBMapper
- Types de données pris en charge pour DynamoDBMapper for Java
- Annotations Java pour DynamoDB
- Paramètres de configuration facultatifs pour DynamoDBMapper
- DynamoDB et verrouillage optimiste avec numéro de version
- Mappage des données arbitraires dans DynamoDB
- Exemples DynamoDBMapper

> **Note**
>
> La classe `DynamoDBMapper` ne vous permet pas de créer, de mettre à jour ou de supprimer des tables. Pour effectuer ces tâches, utilisez plutôt l'interface du kit SDK de bas niveau pour Java.

Le kit SDK pour Java fournit un ensemble de types d'annotations qui vous permettent de mapper vos classes à des tables. Par exemple, imaginons une table `ProductCatalog` ayant `Id` en tant que clé de partition.

```
ProductCatalog(Id, ...)
```

Vous pouvez mapper une classe de votre application client à la table `ProductCatalog`, comme illustré dans le code Java suivant. Cet exemple de code définit un objet Java ancien brut (Plain Old Java Object, POJO) nommé `CatalogItem` qui utilise des annotations pour mapper des champs d'objet à des noms d'attribut DynamoDB.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
```

```
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

Dans le code précédent, l'annotation `@DynamoDBTable` mappe la classe `CatalogItem` à la table `ProductCatalog`. Vous pouvez stocker des instances de classe individuelles en tant qu'éléments dans la table. Dans la définition de classe, l'annotation `@DynamoDBHashKey` mappe l'`Id` de propriété à la clé primaire.

Par défaut, les propriétés de classe mappent avec les mêmes attributs de nom dans la table. Les propriétés `Title` et `ISBN` mappent avec les mêmes attributs de nom dans la table.

L'annotation `@DynamoDBAttribute` est facultative si le nom de l'attribut DynamoDB correspond au nom de la propriété déclarée dans la classe. Si les noms diffèrent, utilisez cette annotation avec le paramètre `attributeName` pour spécifier l'attribut DynamoDB auquel cette propriété correspond.

Dans l'exemple précédent, l'annotation `@DynamoDBAttribute` est ajoutée à chaque propriété afin de garantir que les noms de propriété correspondent exactement aux tables créées au cours d'une étape précédente et d'assurer la cohérence avec les noms d'attributs utilisés dans d'autres exemples de code dans ce guide.

Votre définition de classe peut avoir des propriétés qui ne mappent avec aucun attribut de la table. Vous identifiez ces propriétés en ajoutant l'annotation `@DynamoDBIgnore`. Dans l'exemple précédent, la propriété `SomeProp` est marquée avec l'annotation `@DynamoDBIgnore`. Lorsque vous téléchargez une instance `CatalogItem` vers la table, votre instance `DynamoDBMapper` n'inclut pas de propriété `SomeProp`. En outre, l'outil de mappage ne retourne pas cet attribut lorsque vous récupérez un élément de la table.

Une fois que vous avez défini votre classe de mappage, vous pouvez utiliser des méthodes `DynamoDBMapper` pour écrire une instance de cette classe d'un élément correspondant dans la table `Catalog`. L'exemple de code suivant illustre cette technique.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);
```

```
CatalogItem item = new CatalogItem();
item.setId(102);
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

L'exemple de code suivant montre comment récupérer l'élément et accéder à certains de ses attributs :

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

`DynamoDBMapper` offre un moyen intuitif, naturel d'utiliser des données DynamoDB dans Java. Il offre aussi différentes fonctionnalités intégrées telles que le verrouillage optimiste, les transactions ACID, les valeurs de clé de tri et de clé de partition générées automatiquement et la gestion des versions d'objet.

Classe DynamoDBMapper

La classe `DynamoDBMapper` est le point d'entrée d'Amazon DynamoDB. Elle donne accès à un point de terminaison DynamoDB et vous permet d'accéder à vos données dans différentes tables. Elle vous permet également d'effectuer diverses opérations de création, de lecture, de mise à jour et de suppression (opérations CRUD) sur des éléments, ainsi que d'exécuter des requêtes et analyses sur des tables. Cette classe fournit les méthodes suivantes utiliser DynamoDB.

Pour la documentation Javadoc correspondante, consultez DynamoDBMapper dans la Référence d'API AWS SDK pour Java.

Rubriques

- enregistrer
- charge
- supprimer
- query
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

enregistrer

Enregistre l'objet spécifié dans la table. L'objet que vous voulez enregistrer est le seul paramètre requis pour cette méthode. Vous pouvez fournir des paramètres de configuration facultatifs en utilisant l'objet `DynamoDBMapperConfig`.

Si un élément qui a la même clé primaire n'existe pas, cette méthode crée un nouvel élément dans la table. Si un élément qui a la même clé primaire existe, elle met à jour l'élément existant. Si la clé de partition et la clé de tri sont de type String, et annotées avec `@DynamoDBAutoGeneratedKey`, alors elles reçoivent un identificateur unique universel (UUID) aléatoire en cas de défaut d'initialisation. Les champs de version qui sont annotés avec `@DynamoDBVersionAttribute` seront incrémentés d'un niveau. En outre, si un champ de version est mis à jour ou généré par une clé, l'objet adopté est mis à jour à la suite de l'opération.

Par défaut, seuls les attributs correspondants aux propriétés de classes mappées sont mis à jour. Tous les attributs supplémentaires existant sur un élément ne sont pas affectés. Toutefois, si vous spécifiez `SaveBehavior.CLOBBER`, vous pouvez forcer le remplacement complet de l'élément.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Si vous avez activé le contrôle de version, alors les versions d'élément côté client et côté serveur doivent correspondre. Toutefois, la version n'a pas besoin de correspondre si l'option `SaveBehavior.CLOBBER` est utilisée. Pour plus d'informations sur la gestion des versions, consultez DynamoDB et verrouillage optimiste avec numéro de version.

charge

Récupère un élément dans une table. Vous devez fournir la clé primaire de l'élément que vous voulez récupérer. Vous pouvez fournir des paramètres de configuration facultatifs en utilisant l'objet `DynamoDBMapperConfig`. Par exemple, vous pouvez demander le cas échéant des lectures fortement cohérentes afin de garantir que cette méthode récupère uniquement les dernières valeurs d'élément comme indiqué dans l'instruction Java suivante.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Par défaut, DynamoDB renvoie l'élément ayant des valeurs éventuellement cohérentes. Pour plus d'informations sur le modèle de cohérence éventuelle de DynamoDB, consultez Cohérence en lecture DynamoDB.

supprimer

Supprime un élément de la table. Vous devez communiquer une instance d'objet de la classe mappée.

Si vous avez activé le contrôle de version, alors les versions d'élément côté client et côté serveur doivent correspondre. Toutefois, la version n'a pas besoin de correspondre si l'option `SaveBehavior.CLOBBER` est utilisée. Pour plus d'informations sur la gestion des versions, consultez DynamoDB et verrouillage optimiste avec numéro de version.

query

Interroge une table ou un index secondaire.

Supposons que vous ayez une table, `Reply`, qui stocke des réponses de threads de forum. Chaque objet thread peut avoir zéro ou plusieurs réponses. La clé primaire de la table `Reply` se compose des champs `Id` et `ReplyDateTime`, où `Id` est la clé de partition et `ReplyDateTime` est la clé de tri de la clé primaire.

```
Reply ( Id, ReplyDateTime, ... )
```

Supposons que vous ayez créé un mappage entre une classe `Reply` et la table `Reply` correspondante dans DynamoDB. L'exemple de code Java suivant utilise `DynamoDBMapper` pour trouver toutes les réponses au cours des deux dernières semaines pour un objet thread spécifique.

Example

```
String forumName = "&DDB;";
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

La requête renvoie un ensemble d'objets `Reply`.

Par défaut, la méthode `query` renvoie une collection « avec chargement différé ». Elle ne renvoie initialement qu'une seule page de résultats puis effectue un appel de service pour la page suivante si nécessaire. Pour obtenir tous les éléments correspondants, itérez sur la collection `latestReplies`.

Notez que l'appel de la méthode `size()` sur la collection chargera chaque résultat afin de fournir un décompte précis. Cela peut entraîner la consommation d'un débit alloué élevé et, sur une très grande table, pourrait même épuiser toute la mémoire de votre JVM.

Pour interroger un index, vous devez tout d'abord affecter un modèle à l'index en tant que classe d'outil de mappage. Supposons que la table `Reply` contient un index secondaire global nommé PostedBy-Message-Index. La clé de partition de cet index est `PostedBy`, et la clé de tri est `Message`. La définition de classe d'un élément dans l'index se présenterait comme suit :

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

L'annotation `@DynamoDBTable` indique que cet index est associé à la table `Reply`. L'annotation `@DynamoDBIndexHashKey` désigne la clé de partition (PostedBy) de l'index, et `@DynamoDBIndexRangeKey` désigne la clé de tri (Message) de l'index.

Vous pouvez maintenant utiliser `DynamoDBMapper` pour interroger l'index, en récupérant un sous-ensemble de messages qui ont été publiés par un utilisateur particulier. Il n'est pas nécessaire de spécifier le nom de l'index si vous n'avez pas de mappages contradictoires entre les tables et les index et si les mappages sont déjà établis dans l'outil de mappage. L'outil de mappage déduit en fonction de la clé primaire et de la clé de tri. L'exemple de code suivant interroge un index secondaire global. Étant donné que les index globaux secondaires (gsi) prennent en charge des lectures cohérentes à terme, mais pas des lectures fortement cohérentes, vous devez spécifier `withConsistentRead(false)`.

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
```

```
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);

List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

La requête renvoie un ensemble d'objets `PostedByMessage`.

queryPage

Interroge une table ou un index secondaire, et renvoie une seule page de résultats correspondants. Comme avec la méthode `query`, vous devez spécifier une valeur de clé de partition et un filtre de requête qui est appliqué à l'attribut de clé de tri. Toutefois, `queryPage` renvoie uniquement la première « page » de données, c'est-à-dire la quantité de données pouvant tenir dans 1 Mo

scan

Analyse une table ou un index secondaire entiers. Vous pouvez spécifier le cas échéant un `FilterExpression` pour filtrer l'ensemble de résultats.

Supposons que vous ayez une table, `Reply`, qui stocke des réponses de threads de forum. Chaque objet thread peut avoir zéro ou plusieurs réponses. La clé primaire de la table `Reply` se compose des champs `Id` et `ReplyDateTime`, où `Id` est la clé de partition et `ReplyDateTime` est la clé de tri de la clé primaire.

```
Reply ( Id, ReplyDateTime, ... )
```

Si vous avez mappé une classe Java à la table `Reply`, vous pouvez utiliser le `DynamoDBMapper` pour analyser la table. Par exemple, le code Java suivant analyse l'ensemble de la table `Reply`, en renvoyant uniquement les réponses pour une année donnée.

Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
```

```
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

Par défaut, la méthode `scan` renvoie une collection « avec chargement différé ». Elle ne renvoie initialement qu'une seule page de résultats puis effectue un appel de service pour la page suivante si nécessaire. Pour obtenir tous les éléments correspondants, itérez sur la collection `replies`.

Notez que l'appel de la méthode `size()` sur la collection chargera chaque résultat afin de fournir un décompte précis. Cela peut entraîner la consommation d'un débit alloué élevé et, sur une très grande table, pourrait même épuiser toute la mémoire de votre JVM.

Pour analyser un index, vous devez tout d'abord affecter un modèle à l'index en tant que classe d'outil de mappage. Supposons que la table `Reply` a un index secondaire global nommé `PostedBy-Message-Index`. La clé de partition de cet index est `PostedBy`, et la clé de tri est `Message`. Une classe d'outil de mappage pour cet index est disponible dans la section query. Elle utilise les annotations `@DynamoDBIndexHashKey` et `@DynamoDBIndexRangeKey` pour spécifier la clé de partition et la clé de tri de l'index.

L'exemple de code suivant présente analyse `PostedBy-Message-Index`. Il n'utilise pas un filtre d'analyse. Ainsi, tous les éléments dans l'index vous sont renvoyés.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Analyse une table ou un index secondaire, et renvoie une seule page de résultats correspondants. Comme avec la méthode `scan`, vous pouvez spécifier le cas échéant un `FilterExpression` pour filtrer le jeu de résultats. Toutefois, `scanPage` renvoie uniquement la première « page » de données, c'est-à-dire la quantité de données pouvant tenir dans 1 Mo.

parallelScan

Effectue une analyse parallèle d'une table ou d'un index secondaire entiers. Vous spécifiez un certain nombre de segments logiques pour la table, avec une expression d'analyse pour filtrer les résultats. Le `parallelScan` répartit la tâche d'analyse entre plusieurs instances de travail, une pour chaque segment logique ; les instances de travail traitent les données en parallèle et renvoient les résultats.

L'exemple de code Java suivant effectue une analyse parallèle sur la table `Product`.

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);

List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

batchSave

Enregistre des objets dans une ou plusieurs tables à l'aide d'un ou de plusieurs appels à la méthode `AmazonDynamoDB.batchWriteItem`. Cette méthode n'offre pas de garanties de transaction.

Le code Java enregistre deux éléments (livres) dans la table `ProductCatalog`.

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");

mapper.batchSave(Arrays.asList(book1, book2));
```

batchLoad

Récupère plusieurs éléments d'une ou de plusieurs tables en utilisant leurs clés primaires.

Le code Java suivant récupère deux éléments dans deux tables différentes.

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

batchDelete

Supprime des objets d'une ou de plusieurs tables à l'aide d'un ou de plusieurs appels à la méthode `AmazonDynamoDB.batchWriteItem`. Cette méthode n'offre pas de garanties de transaction.

Le code Java supprime deux éléments (livres) dans la table `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Enregistre et supprime des objets dans une ou plusieurs des tables à l'aide d'un ou de plusieurs appels à la méthode `AmazonDynamoDB.batchWriteItem`. Cette méthode ne fournit pas des garanties de transactions et ne prend pas en charge le contrôle de version (suppressions ou insertions conditionnelles).

Le code Java suivant écrit un nouvel élément dans la table `Forum`, écrit un nouvel élément dans la table `Thread` et supprime un élément de la table `ProductCatalog`.

```
// Create a Forum item to save
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
```

```
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");

// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

transactionWrite

Enregistre et supprime des objets dans une ou plusieurs des tables à l'aide d'un appel à la méthode `AmazonDynamoDB.transactWriteItems`.

Pour obtenir la listes des exceptions spécifiques aux transactions, consultez TransactWriteItems errors (erreurs TransactWriteItems).

Pour plus d'informations sur les transactions DynamoDB et les garanties d'atomicité, de cohérence, d'isolation et de durabilité (ACID) offertes, consultez Transactions d'Amazon DynamoDB.

> **Note**
>
> Cette méthode ne prend pas en charge :
>
> - DynamoDBMapperConfig.SaveBehavior.

Le code Java suivant écrit un nouvel élément dans chacune des tables `Forum` et `Thread`, par voie de transaction.

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);
```

```
TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

Charge des objets d'une ou de plusieurs tables à l'aide d'un appel à la méthode `AmazonDynamoDB.transactGetItems`.

Pour obtenir la liste des exceptions spécifiques aux transactions, consultez TransactWriteItems errors (erreurs TransactWriteItems).

Pour plus d'informations sur les transactions DynamoDB et les garanties d'atomicité, de cohérence, d'isolation et de durabilité (ACID) offertes, consultez Transactions d'Amazon DynamoDB.

L'extrait de code Java suivent charge un élément à partir de chacune des tables `Forum` et `Thread`, par voie de transaction.

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

count

Évalue l'expression d'analyse spécifiée et renvoie le nombre d'éléments correspondants. Aucun élément de données n'est renvoyé.

generateCreateTableRequest

Analyse une classe POJO représentant une table DynamoDB, et renvoie une `CreateTableRequest` pour cette table.

createS3Link

Crée un lien vers un objet dans Amazon S3. Vous devez spécifier un nom de compartiment et un nom de clé identifiant l'objet de manière unique dans le compartiment.

Pour utiliser `createS3Link`, votre classe d'outil de mappage doit définir des méthodes getter et setter. L'exemple de code suivant illustre cela en ajoutant un nouvel attribut et des méthodes getter/setter à la classe `CatalogItem` :

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...

    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

Le code Java suivant définit un nouvel élément à écrire dans la table `Product`. L'élément inclut un lien vers une image de produit ; les données d'image sont chargées dans Amazon S3.

```
CatalogItem item = new CatalogItem();

item.setId(150);
item.setTitle("Book 150 Title");

String amzn-s3-demo-bucket = "amzn-s3-demo-bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(amzn-s3-demo-bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

La classe `S3Link` fournit de nombreuses autres méthodes pour la manipulation d'objets dans Amazon S3. Pour plus d'informations, consultez Javadocs pour `S3Link`.

getS3ClientCache

Renvoie le `S3ClientCache` sous-jacent pour l'accès à Amazon S3. Un `S3ClientCache` est une carte intelligente pour des objets `AmazonS3Client`. Si vous avez plusieurs clients, un `S3ClientCache` peut vous aider à organiser les clients par région AWS, et créer de nouveaux clients Amazon S3 à la demande.

Types de données pris en charge pour DynamoDBMapper for Java

Cette section décrit les types de données Java primitifs, les collections et les types de données arbitraires pris en charge dans Amazon DynamoDB.

Amazon DynamoDB prend en charge les types de données Java primitifs et les classes wrapper primitives suivants.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (en tant que chaîne de précision à la milliseconde ISO_8601, passée sur UTC)
- `Calendar` (en tant que chaîne de précision à la milliseconde ISO_8601, passée sur UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> **Note**
>
> - Pour plus d'informations sur les règles de dénomination de DynamoDB et les différents types de données pris en charge, consultez Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

- Les valeurs binaires vides sont prises en charge par DynamoDBMapper.
- AWS SDK for Java 2.x prend en charge les valeurs String vides.

  Dans le kit SDK AWS pour Java 1.x, DynamoDBMapper prend en charge la lecture des valeurs d'attribut String vides, mais pas leur écriture, car ces attributs sont supprimés de la demande.

DynamoDB prend en charge les types de collections Java Set (ensemble), List (liste) et Map (mappage). Le tableau suivant résume la manière dont ces types Java mappent aux types DynamoDB.

Type Java	Type DynamoDB
Tous les types de numéro	`N` (type Number)
Chaînes	`S` (type String)
Booléen	`BOOL` (type booléen), 0 ou 1.
ByteBuffer	`B` (type Binary)
Date	`S` (type String). Les valeurs Date sont stockées comme chaînes formatées ISO-8601.
Types de collections Set	Type `SS` (string set), type `NS` (number set) ou type `BS` (binary set).

L'interface `DynamoDBTypeConverter` vous permet de mapper vos propres types de données arbitraires à un type de données que DynamoDB prend en charge en mode natif. Pour plus d'informations, consultez Mappage des données arbitraires dans DynamoDB.

Annotations Java pour DynamoDB

Cette section décrit les annotations disponibles pour le mappage de vos classes et propriétés à des tables et attributs dans Amazon DynamoDB.

Pour la documentation Javadoc correspondante, consultez Annotation Types Summary dans la Référence d'API AWS SDK pour Java.

> **Note**
>
> Dans les annotations suivantes, seules `DynamoDBTable` et `DynamoDBHashKey` sont obligatoires.

Rubriques

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

Mappe une propriété avec un attribut de table. Par défaut, chaque propriété de classe mappe un attribut d'élément avec le même nom. Toutefois, si les noms ne sont pas les mêmes, vous pouvez utiliser cette annotation pour mapper une propriété avec l'attribut. Dans l'extrait Java suivant, l'`DynamoDBAttribute` mappe la propriété `BookAuthors` avec le nom d'attribut `Authors` de la table.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

Le `DynamoDBMapper` utilise `Authors` en tant que nom d'attribut lors de l'enregistrement de l'objet vers la table.

DynamoDBAutoGeneratedKey

Marque une propriété de clé de tri ou de clé de partition comme étant générée automatiquement. `DynamoDBMapper` génère un UUID aléatoire lors de l'enregistrement de ces attributs. Seules les propriétés String peuvent être marquées en tant que clés générées automatiquement.

L'exemple suivant illustre l'utilisation clés générées automatiquement.

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);

        System.out.println("Object was saved with id " + obj.getId());
    }
}
```

DynamoDBAutoGeneratedTimestamp

Génère automatiquement un horodatage.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
```

```
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

(Facultatif) La stratégie de génération automatique peut être définie en fournissant un attribut de stratégie. L'argument par défaut est `ALWAYS`.

DynamoDBDocument

Indique qu'une classe peut être sérialisée sous forme de document Amazon DynamoDB.

Par exemple, supposons que vous voulez mapper un document JSON à un attribut DynamoDB de type Map (`M`). L'exemple de code suivant définit un élément contenant un attribut imbriqué de type de Map.

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
        public void setRearView(String rearView) { this.rearView = rearView; }
```

```
        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

    }
}
```

Vous pouvez alors enregistrer un nouvel élément `ProductCatalog` avec `Pictures`, comme illustré dans l'exemple suivant.

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

L'élément `ProductCatalog` qui en résulte se présente comme suit (au format JSON) :

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

DynamoDBHashKey

Mappe une propriété de classe avec la clé de partition de la table. La propriété doit être un des types binaire, numéro ou chaîne scalaire. La propriété ne peut pas être de type collection.

Supposons que vous ayez une table, `ProductCatalog`, qui a un `Id` en tant que clé primaire. Le code Java suivant définit une classe `CatalogItem` et mappe sa propriété `Id` à la clé primaire de la table `ProductCatalog` à l'aide de la balise `@DynamoDBHashKey`.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

DynamoDBIgnore

Indique à l'instance `DynamoDBMapper` que la propriété associée doit être ignorée. Lors de l'enregistrement de données dans la table, `DynamoDBMapper` ne sauvegarde pas cette propriété dans la table.

S'applique à la méthode getter ou au champ de classe pour une propriété non modélisée.
Si l'annotation est appliquée directement au champ de classe, les méthodes getter et setter correspondantes doivent être déclarées dans la même classe.

DynamoDBIndexHashKey

Mappe une propriété de classe à la clé de partition d'un index secondaire global. La propriété doit être un des types binaire, numéro ou chaîne scalaire. La propriété ne peut pas être de type collection.

Utilisez cette annotation si vous devez interroger (`Query`) un index secondaire global. Vous devez spécifier le nom d'index (`globalSecondaryIndexName`). Si le nom de la propriété de classe est différent de la clé de partition d'index, vous devez également spécifier le nom de cet attribut d'index (`attributeName`).

DynamoDBIndexRangeKey

Mappe une propriété de classe à la clé de tri d'un index secondaire global ou local. La propriété doit être un des types binaire, numéro ou chaîne scalaire. La propriété ne peut pas être de type collection.

Utilisez cette annotation si vous devez interroger (`Query`) un index secondaire local ou global, et souhaitez affiner vos résultats à l'aide de la clé de tri d'index. Vous devez spécifier le nom d'index (soit `globalSecondaryIndexName`, soit `localSecondaryIndexName`). Si le nom de la propriété

de classe est différent de la clé de tri d'index, vous devez également spécifier le nom de cet attribut d'index (`attributeName`).

DynamoDBRangeKey

Mappe une propriété de classe avec la clé de tri de la table. La propriété doit être un des types binaire, numéro ou chaîne scalaire. Elle ne peut pas être de type collection.

Si la clé primaire est composite (clé de partition et clé de tri), vous pouvez utiliser cette balise pour mapper votre champ de classe avec la clé de tri. Par exemple, supposons que vous ayez une table `Reply` qui stocke les réponses pour les threads de forum. Chaque thread peut avoir plusieurs réponses. Ainsi, la clé primaire de cette table est à la fois `ThreadId` et `ReplyDateTime`. La clé de partition est `ThreadId` et la clé de tri est `ReplyDateTime`.

L'extrait de code Java suivant définit une classe `Reply` et la mappe à la table `Reply`. Il utilise les deux balises, `@DynamoDBHashKey` et `@DynamoDBRangeKey`, pour identifier les propriétés de classe qui sont mappées avec la clé primaire.

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }

   // Additional properties go here.
}
```

DynamoDBTable

Identifie la table cible dans DynamoDB. Par exemple, l'extrait de code Java suivant définit une classe `Developer` et la mappe à la table `People` dans DynamoDB.

```
@DynamoDBTable(tableName="People")
```

```
public class Developer { ...}
```

L'annotation `@DynamoDBTable` peut être héritée. Toute nouvelle classe qui hérite de la classe `Developer` se mappe également à la table `People`. Par exemple, supposons que vous génériez une classe `Lead` qui hérite de la classe `Developer`. Étant donné que vous avez mappé la classe `Developer` à la table `People`, les objets de la classe `Lead` sont également stockés dans la même table.

La `@DynamoDBTable` peut également être remplacée. Toute nouvelle classe qui hérite de la classe `Developer` par défaut se mappe à la même table `People`. Toutefois, vous pouvez substituer ce mappage par défaut. Par exemple, si vous créez une classe qui hérite de la classe `Developer`, vous pouvez explicitement la mapper à une autre table en ajoutant l'annotation `@DynamoDBTable` comme illustré dans l'extrait de code Java suivant.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

Une annotation permettant d'indiquer qu'une propriété utilise un convertisseur personnalisé. Peut être indiquée sur une annotation définie par l'utilisateur afin de transmettre des propriétés à `DynamoDBTypeConverter`.

L'interface `DynamoDBTypeConverter` vous permet de mapper vos propres types de données arbitraires à un type de données que DynamoDB prend en charge en mode natif. Pour plus d'informations, consultez Mappage des données arbitraires dans DynamoDB.

DynamoDBTyped

Une annotation permettant de remplacer la liaison du type d'attribut standard. Les types standard ne nécessitent pas l'annotation si la liaison d'attribut par défaut est appliquée pour ce type.

DynamoDBVersionAttribute

Identifie une propriété de classe pour stocker un numéro de version de verrouillage optimiste. `DynamoDBMapper` affecte un numéro de version à cette propriété lorsqu'il enregistre un nouvel élément et l'augmente chaque fois que vous mettez à jour l'élément. Seuls les types scalaires de numéros sont pris en charge. Pour plus d'informations sur les types de données, consultez Types de données, Pour plus d'informations sur la gestion des versions, consultez DynamoDB et verrouillage optimiste avec numéro de version.

Paramètres de configuration facultatifs pour DynamoDBMapper

Lorsque vous créez une instance de `DynamoDBMapper`, elle a certains comportements par défaut ; vous pouvez remplacer ces valeurs par défaut en utilisant la classe `DynamoDBMapperConfig`.

L'extrait de code suivant crée un `DynamoDBMapper` avec des paramètres personnalisés :

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

 .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
    .build();

DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

Pour plus d'informations, consultez DynamoDBMapperConfig dans la Référence d'API Amazon DynamoDBAWS SDK pour Java.

Vous pouvez utiliser les arguments suivants pour une instance de `DynamoDBMapperConfig` :

- Une valeur d'énumération `DynamoDBMapperConfig.ConsistentReads` :
  - `EVENTUAL` – L'instance du mappeur utilise une demande de lecture éventuellement cohérente.
  - `CONSISTENT` – L'instance du mappeur utilise une demande de lecture fortement cohérente. Vous pouvez utiliser ce paramètre facultatif avec des opérations `load`, `query` ou `scan`. Les lectures fortement cohérentes ont des implications pour les performances et la facturation. Pour plus d'informations, consultez la page détaillée du produit.

  Si vous ne spécifiez pas un paramètre de cohérence de lecture pour votre instance d'outil de mappage, la valeur par défaut est `EVENTUAL`.

> **Note**
>
> Cette valeur est appliquée dans les opérations `query`, `querypage`, `load` et `batch load` de DynamoDBMapper.

- Valeur d'énumération `DynamoDBMapperConfig.PaginationLoadingStrategy` – Contrôle la façon dont l'instance de mappeur traite une liste paginée de données, telle que les résultats d'une opération `query` ou `scan` :
  - `LAZY_LOADING` – L'instance de mappeur charge les données quand cela est possible, et conserve tous les résultats chargés en mémoire.
  - `EAGER_LOADING` – L'instance de mappeur charge les données dès que la liste est initialisée.
  - `ITERATION_ONLY` – Vous ne pouvez utiliser qu'un itérateur pour lire la liste. Pendant l'itération, la liste efface tous les résultats précédents avant de charger la page suivante, pour que la liste conserve en mémoire au maximum une page des résultats chargés. Cela signifie également que la liste ne peut être itérée qu'à une seule reprise. Cette stratégie est recommandée lors de la gestion d'éléments importants, afin de réduire les frais généraux de mémoire.

  Si vous ne spécifiez pas une stratégie de chargement de pagination pour votre instance d'outil de mappage, la valeur par défaut est `LAZY_LOADING`.
- Une valeur d'énumération `DynamoDBMapperConfig.SaveBehavior` - Spécifie comment l'instance d'outil de mappage devrait traiter les attributs lors des opérations d'enregistrement :
  - `UPDATE` – Lors d'une opération d'enregistrement, tous les attributs modélisés sont mis à jour, et les attributs non modélisés ne sont pas affectés. Les types de numéros primitifs (octets, int, long) sont définis sur 0. Les types d'objets sont définis sur null.
  - `CLOBBER` – Efface et remplace tous les attributs, y compris non modélisés, lors d'une opération d'enregistrement. Pour cela, il faut supprimer l'élément et le recréer. Les contraintes de champ avec version sont également ignorées.

  Si vous ne spécifiez pas le comportement de sauvegarde pour votre instance d'outil de mappage, la valeur par défaut est `UPDATE`.

  > **Note**
  >
  > Les opérations transactionnelles DynamoDBMapper ne prennent pas en charge l'énumération `DynamoDBMapperConfig.SaveBehavior`.
- Objet `DynamoDBMapperConfig.TableNameOverride` – Indique à l'instance de mappeur d'ignorer le nom de table spécifié par l'annotation `DynamoDBTable` d'une classe, et d'utiliser plutôt un autre nom de table que vous fournissez. Cela est utile lors du partitionnement de vos données dans plusieurs tables au moment de l'exécution.

Vous pouvez remplacer l'objet de configuration par défaut pour `DynamoDBMapper` par opération, en fonction des besoins.

DynamoDB et verrouillage optimiste avec numéro de version

Le verrouillage optimiste est une politique visant à garantir que l'élément côté client que vous mettez à jour (ou supprimez) est identique à l'élément dans Amazon DynamoDB. Si vous utilisez cette stratégie, les écritures de votre base de données ne peuvent pas être remplacées par les écritures d'autres, et vice versa.

Avec le verrouillage optimiste, chaque élément possède un attribut qui agit comme un numéro de version. Si vous récupérez un élément à partir d'une table, l'application enregistre le numéro de version de cet élément. Vous pouvez mettre à jour l'élément, mais uniquement si le numéro de version sur le côté serveur n'a pas changé. Si les versions ne correspondent pas, cela signifie que quelqu'un d'autre a modifié cet élément avant vous. La tentative de mise à jour échoue car vous possédez une version obsolète de l'élément. Si cela se produit, essayez à nouveau en récupérant l'élément, puis en tentant de le mettre à jour. Le verrouillage optimiste vous empêche de remplacer accidentellement les modifications effectuées par les autres. Il empêche également que vos modifications soient remplacées.

Bien que vous puissiez mettre en œuvre votre propre stratégie de verrouillage optimiste, AWS SDK pour Java fournit l'annotation `@DynamoDBVersionAttribute`. Dans la classe de mappage pour votre table, vous indiquez une propriété pour stocker le numéro de version et vous la marquez à l'aide de cette annotation. Lorsque vous enregistrez un objet, l'élément correspondant dans la table DynamoDB aura un attribut qui stocke le numéro de version. Le `DynamoDBMapper` attribue un numéro de version lors de l'enregistrement initial de l'objet, et il augmente automatiquement le numéro de version chaque fois que vous mettez à jour l'élément. Vos demandes de mise à jour ou de suppression aboutissent uniquement si la version d'objet côté client correspond au numéro de version de l'élément correspondant dans la table DynamoDB.

`ConditionalCheckFailedException` est émis si :

- Vous utilisez le verrouillage optimiste avec `@DynamoDBVersionAttribute` et la valeur de version sur le serveur est différente de la valeur côté client.
- Vous spécifiez vos propres contraintes conditionnelles tout en enregistrant des données à l'aide de `DynamoDBMapper` avec `DynamoDBSaveExpression` et ces contraintes ont échoué.

> **Note**
>
> - Les tables globales DynamoDB utilisent un rapprochement « last writer wins » (dernière version valide) entre des mises à jour concomitantes. Si vous utilisez des tables globales, la stratégie « last writer wins » (dernière version valide) s'applique. Dans ce cas, la stratégie de verrouillage ne fonctionne pas comme prévu.
> - Les opérations d'écriture transactionnelle `DynamoDBMapper` ne prennent pas en charge les expressions d'annotation et de condition `@DynamoDBVersionAttribute` sur le même objet. Au sein d'une écriture transactionnelle, si un objet est annoté avec `@DynamoDBVersionAttribute` et possède également une expression de condition, alors une exception SdkClientException est émise.

Par exemple, le code Java suivant définit une classe `CatalogItem` qui possède plusieurs propriétés. La propriété `Version` est balisée avec l'annotation `@DynamoDBVersionAttribute`.

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}
```

```
    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

Vous pouvez appliquer l'annotation `@DynamoDBVersionAttribute` aux types Nullable proposés par les classes de wrappers primitifs qui fournissent un type Nullable, par exemple `Long` et `Integer`.

Le verrouillage optimiste a l'impact suivant sur ces méthodes `DynamoDBMapper` :

- `save` – Pour un nouvel élément, le `DynamoDBMapper` attribue un numéro de version initial 1. Si vous récupérez un élément, que vous mettez à jour une ou plusieurs de ses propriétés et que vous tentez d'enregistrer les modifications, l'opération de sauvegarde n'aboutit que si les numéros de version côté client et côté serveur correspondent. Le `DynamoDBMapper` augmente le numéro de version automatiquement.
- `delete` – La méthode `delete` prend un objet en tant que paramètre, et le `DynamoDBMapper` effectue une vérification de version avant de supprimer l'élément. Le contrôle de version peut être désactivé si `DynamoDBMapperConfig.SaveBehavior.CLOBBER` est spécifié dans la demande.

  L'implémentation interne de verrouillage optimiste au sein de `DynamoDBMapper` utilise la prise en charge de suppression et de mise à jour conditionnelles fournies par DynamoDB.
- `transactionWrite` —
  - `Put` – Pour un nouvel élément, le `DynamoDBMapper` attribue un numéro de version initial 1. Si vous récupérez un élément, que vous mettez à jour une ou plusieurs de ses propriétés et que vous tentez d'enregistrer les modifications, l'opération « Put » n'aboutit que si les numéros de version côté client et côté serveur correspondent. Le `DynamoDBMapper` augmente le numéro de version automatiquement.
  - `Update` – Pour un nouvel élément, le `DynamoDBMapper` attribue un numéro de version initial 1. Si vous récupérez un élément, que vous mettez à jour une ou plusieurs de ses propriétés et que vous tentez d'enregistrer les modifications, l'opération de mise à jour n'aboutit que si les

numéros de version côté client et côté serveur correspondent. Le `DynamoDBMapper` augmente le numéro de version automatiquement.

- `Delete` – Le `DynamoDBMapper` effectue une vérification de version avant de supprimer l'élément. L'opération de suppression n'aboutit que si les numéros de version côté client et côté serveur correspondent.
- `ConditionCheck` – L'annotation `@DynamoDBVersionAttribute` n'est pas prise en charge pour les opérations `ConditionCheck`. Une exception SdkClientException est émise lorsqu'un élément `ConditionCheck` est annoté avec `@DynamoDBVersionAttribute`.

Désactivation du verrouillage optimiste

Pour désactiver le verrouillage optimiste, vous pouvez changer la valeur d'énumération `DynamoDBMapperConfig.SaveBehavior` de `UPDATE` à `CLOBBER`. Vous pouvez le faire en créant une instance `DynamoDBMapperConfig` qui ignore la vérification de version et utilise cette instance pour toutes vos demandes. Pour plus d'informations sur les paramètres `DynamoDBMapperConfig.SaveBehavior` et autres paramètres facultatifs `DynamoDBMapper`, consultez Paramètres de configuration facultatifs pour DynamoDBMapper .

Vous pouvez également définir le comportement de verrouillage pour une opération spécifique uniquement. Par exemple, l'extrait de Java suivant utilise le `DynamoDBMapper` pour enregistrer un élément de catalogue. Il spécifie `DynamoDBMapperConfig.SaveBehavior` en ajoutant le paramètre `DynamoDBMapperConfig` facultatif à la méthode `save`.

> **Note**
>
> La méthode transactionWrite ne prend pas en charge la configuration DynamoDBMapperConfig.SaveBehavior. La désactivation du verrouillage optimiste pour transactionWrite n'est pas prise en charge.

Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
```

```
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

Mappage des données arbitraires dans DynamoDB

En plus des types Java pris en charge (voir Types de données pris en charge pour DynamoDBMapper for Java), vous pouvez utiliser des types dans votre application pour lesquels il n'y a pas de mappage direct aux types Amazon DynamoDB. Pour mapper ces types, vous devez fournir une implémentation qui convertit votre type complexe en un type pris en charge par DynamoDB et inversement, et annoter la méthode accesseur de type complexe en utilisant l'annotation `@DynamoDBTypeConverted`. Le code de convertisseur transforme les données lorsque des objets sont enregistrés ou chargés. Il est également utilisé pour toutes les opérations qui utilisent des types complexes. Notez que, lors de la comparaison de données pendant des opérations d'interrogation et d'analyse, les comparaisons sont faites par rapport aux données stockées dans DynamoDB.

Par exemple, envisagez la classe `CatalogItem` suivante qui définit une propriété, `Dimension`, qui est de `DimensionType`. Cette propriété stocke les dimensions de l'élément, telles que la hauteur, la largeur et l'épaisseur. Supposons que vous décidez de stocker ces dimensions d'élément sous la forme d'une chaîne (par exemple, 8.5x11x.05) dans DynamoDB. L'exemple suivant fournit un code de convertisseur qui convertit l'objet `DimensionType` en une chaîne et une chaîne en `DimensionType`.

> **Note**
>
> Cet exemple de code part du principe que vous avez déjà chargé des données dans DynamoDB pour votre compte en suivant les instructions de la section Création de tables et chargement de données pour des exemples de code dans DynamoDB.
> Pour obtenir les instructions pas à pas afin d'exécuter l'exemple suivant, consultez Exemples de code Java.

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;
```

```
    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Book info: " + "\n" + bookRetrieved);

        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
```

```
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
```

```
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }

        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }

        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }
```

```
    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
 DimensionType> {

        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
                    dimension = String.format("%s x %s x %s",
 itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {

            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
}
```

Exemples DynamoDBMapper

Le kit AWS SDK fournit une classe `DynamoDBMapper` qui vous permet de mapper vos classes côté client à des tables DynamoDB. Pour utiliser `DynamoDBMapper`, vous définissez la relation entre éléments d'une table DynamoDB et leurs instances d'objet correspondantes dans votre code. La classe `DynamoDBMapper` vous permet d'effectuer diverses opérations de création, de lecture, de mise à jour et de suppression (opérations CRUD) sur des éléments, ainsi que d'exécuter des requêtes et analyses sur des tables.

Pour en savoir plus sur l'utilisation de `DynamoDBMapper`, consultez DynamoDB Examples Using the AWS SDK for Java dans le Guide du développeur AWS SDK for Java 1.x.

Java 2.x : client amélioré DynamoDB

Le client amélioré DynamoDB est une bibliothèque de haut niveau qui fait partie du kit AWS SDK pour Java version 2 (v2). Il offre un moyen simple de mapper des classes côté client à des tables DynamoDB. Vous définissez les relations entre des tables et leurs classes de modèle correspondantes dans votre code. Après avoir défini ces relations, vous pouvez effectuer de manière intuitive diverses opérations de création, de lecture, de mise à jour ou de suppression (CRUD) sur des tables ou des éléments dans DynamoDB.

Pour plus d'informations sur l'utilisation du client amélioré avec DynamoDB, consultez Utilisation du client amélioré DynamoDB dans le kit AWS SDK pour Java version 2.x.

Utilisation du modèle de document .NET dans DynamoDB

AWS SDK pour .NET fournit des classes de modèle de document qui encapsulent certaines opérations Amazon DynamoDB de bas niveau pour simplifier davantage votre codage. Dans le modèle de document, les classes principales sont `Table` et `Document`. La classe `Table` fournit des méthodes d'opération de données telles que `PutItem`, `GetItem` et `DeleteItem`. Elle fournit aussi les méthodes `Query` et `Scan`. La classe `Document` représente un seul élément d'une table.

Les classes de modèle de document précédentes sont disponibles dans l'espace de noms `Amazon.DynamoDBv2.DocumentModel`.

> **Note**
>
> Vous ne pouvez pas utiliser les classes de modèle de document pour créer, mettre à jour et supprimer des tables. Cependant, le modèle de document ne prend pas en charge les opérations de données les plus courantes.

Rubriques

- Types de données pris en charge

Types de données pris en charge

Le modèle de document prend en charge un ensemble de types de données .NET primitifs et de types de données de collections. Le modèle prend en charge les types de données primitifs suivants.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Le tableau suivant résume le mappage des types .NET précédents aux types DynamoDB.

Type primitif .NET	Type DynamoDB
Tous les types de numéro	`N` (type Number)
Tous types de chaînes	`S` (type String)
MemoryStream, byte[]	`B` (type Binary)

Type primitif .NET	Type DynamoDB
bool	N (type de nombre). 0 représente false et 1 représente true.
DateTime	S (type String). Les valeurs DateTime sont stockées comme chaînes de format ISO-8601.
Guid	S (type String).
Types de collection (liste, HashSet et tableau)	Type BS (ensemble de binaires), type SS (ensemble de chaînes) ou type NS (ensemble de nombres).

AWS SDK pour .NET définit les types de mappages (boolean (booléen), null (nul), list (liste) et map (mappage) de DynamoDB à l'API de modèle de document .NET :

- Utilisez `DynamoDBBool` pour le type booléen.
- Utilisez `DynamoDBNull` pour le type null.
- Utilisez `DynamoDBList` pour le type list.
- Utilisez `Document` pour le type map.

> **Note**
>
> - Les valeurs binaires vides sont prises en charge.
> - La lecture des valeurs de chaîne vides est prise en charge. Les valeurs d'attribut de chaîne vides sont prises en charge dans les valeurs d'attribut de type de chaîne Set lors de l'écriture dans DynamoDB. Les valeurs d'attribut de chaîne vides de type String et les valeurs de chaîne vides contenues dans le type List ou Map sont supprimées des demandes d'écriture.

Utilisation du modèle de persistance des objets .NET et de DynamoDB

AWS SDK pour .NET fournit un modèle de persistance des objets qui vous permet de mapper vos classes côté client aux tables Amazon DynamoDB. Ensuite, chaque instance d'objet est mappée

à un élément des tables correspondantes. Pour enregistrer vos objets côté client dans les tables, le modèle de persistance des objets fournit la classe `DynamoDBContext`, un point d'entrée dans DynamoDB. Cette classe fournit une connexion à DynamoDB, et vous permet d'accéder aux tables, d'effectuer diverses opérations CRUD ainsi que d'exécuter des requêtes.

Le modèle de persistance des objets fournit un ensemble d'attributs permettant de mapper les classes côté client aux tables et properties/fields aux attributs des tables.

> **Note**
>
> Le modèle de persistance des objets n'autorise pas une API à créer, mettre à jour ou supprimer des tables. Il fournit uniquement des opérations sur les données. Vous ne pouvez utiliser que l'API de AWS SDK pour .NET bas niveau pour créer, mettre à jour et supprimer des tables.

L'exemple suivant montre le fonctionnement du modèle de persistance des objets. Il commence par la table `ProductCatalog`. Sa clé primaire est `Id`.

```
ProductCatalog(Id, ...)
```

Supposons que vous avec une classe `Book` avec les propriétés `Title`, `ISBN` et `Authors`. Vous pouvez mapper la classe `Book` à la table `ProductCatalog` en ajoutant les attributs définis par le modèle de persistance des objets, comme illustré dans l'exemple de code C# suivant.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
```

```
    public string CoverPage { get; set; }
}
```

Dans l'exemple précédent, l'attribut `DynamoDBTable` mappe la classe `Book` à la table `ProductCatalog`.

Le modèle de persistance des objets prend en charge les mappages explicite et par défaut entre les propriétés de classe et les attributs de table.

- Mappage explicite – Pour mapper une propriété à une clé primaire, vous devez utiliser les attributs de modèle de persistance des objets `DynamoDBHashKey` et `DynamoDBRangeKey`. En outre, pour les attributs de clé non primaire, si un nom de propriété de votre classe et l'attribut de table correspondant auquel vous voulez le mapper ne sont pas identiques, vous devez définir le mappage en ajoutant explicitement l'attribut `DynamoDBProperty`.

  Dans l'exemple précédent, la propriété `Id` mappe à la clé primaire du même nom, et la propriété `BookAuthors` mappe à l'attribut `Authors` dans la table `ProductCatalog`.
- Mappage par défaut – Par défaut, le modèle de persistance des objets mappe les propriétés de classe aux attributs du même nom dans la table.

  Dans l'exemple précédent, les propriétés `Title` et `ISBN` mappent aux attributs du même nom dans la table `ProductCatalog`.

Vous n'avez pas à mapper chaque propriété de classe. Vous identifiez ces propriétés en ajoutant l'attribut `DynamoDBIgnore`. Lorsque vous enregistrez une instance `Book` dans la table, le `DynamoDBContext` n'inclut pas la propriété `CoverPage`. Il ne renvoie pas non plus cette propriété lorsque vous récupérez l'instance livre.

Vous pouvez mapper des propriétés de types primitifs .NET tels que int et string. Vous pouvez également mapper n'importe quels types de données arbitraires tant que vous fournissez un convertisseur approprié pour mapper les données arbitraires à l'un des types DynamoDB. Pour en savoir plus sur le mappage de types arbitraires, consultez Mappage de données arbitraires avec DynamoDB à l'aide AWS SDK pour .NET du modèle de persistance des objets.

Le modèle de persistance des objets prend en charge le verrouillage optimiste. Au cours d'une opération de mise à jour, cela garantit que vous disposez de la dernière copie de l'élément que vous êtes sur le point de mettre à jour. Pour de plus amples informations, veuillez consulter Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET.

Pour plus d'informations, consultez les rubriques ci-dessous.

Rubriques

- Types de données pris en charge
- Attributs DynamoDB du modèle de persistance des objets .NET
- DBContext Classe Dynamo issue du modèle de persistance des objets .NET
- Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET
- Mappage de données arbitraires avec DynamoDB à l'aide AWS SDK pour .NET du modèle de persistance des objets

Types de données pris en charge

Le modèle de persistance des objets prend en charge un ensemble de types de données .NET primitifs, de collections et de types de données arbitraires. Le modèle prend en charge les types de données primitifs suivants.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Le modèle de persistance des objets prend également en charge les types de collection .NET. `DynamoDBContext`est capable de convertir des types de collection concrets et de simples objets CLR ordinaires (POCOs).

Le tableau suivant résume le mappage des types .NET précédents aux types DynamoDB.

Type primitif .NET	Type DynamoDB
Tous les types de numéro	`N` (type Number)
Tous types de chaînes	`S` (type String)
MemoryStream, octet []	`B` (type Binary)
bool	`N` (type de nombre). `0` représente false et `1` représente true.
Types de collections	Type `BS` (ensemble de binaires), type `SS` (ensemble de chaînes) ou type `NS` (ensemble de nombres).
DateTime	`S` (type String). Les valeurs `DateTime` sont stockées comme chaînes de format ISO-8601.

Le modèle de persistance des objets prend également en charge les types de données arbitraires. Toutefois, vous devez fournir un code de convertisseur pour mapper les types complexes aux types DynamoDB.

> **Note**
>
> - Les valeurs binaires vides sont prises en charge.
> - La lecture des valeurs de chaîne vides est prise en charge. Les valeurs d'attribut de chaîne vides sont prises en charge dans les valeurs d'attribut de type de chaîne Set lors de l'écriture dans DynamoDB. Les valeurs d'attribut de chaîne vides de type String et les valeurs de chaîne vides contenues dans le type List ou Map sont supprimées des demandes d'écriture.

Attributs DynamoDB du modèle de persistance des objets .NET

Cette section décrit les attributs qu'offre le modèle de persistance des objets afin que vous puissiez mapper vos classes et propriétés aux tables et attributs DynamoDB.

> **Note**
>
> Dans les attributs suivants, seuls les attributs `DynamoDBTable` et `DynamoDBHashKey` sont obligatoires.

Dynamo DBGlobal SecondaryIndexHashKey

Mappe une propriété de classe à la clé de partition d'un index secondaire global. Utilisez cet attribut si vous devez interroger (`Query`) un index secondaire global.

Dynamo DBGlobal SecondaryIndexRangeKey

Mappe une propriété de classe à la clé de tri d'un index secondaire global. Utilisez cet attribut si vous devez interroger (`Query`) un index secondaire global, et souhaitez affiner vos résultats à l'aide de la clé de tri d'index.

Clé Dynamo DBHash

Mappe une propriété de classe à la clé de partition de la clé primaire de la table. Les attributs de clé primaire ne peuvent pas être de type collection.

L'exemple de code C# suivant mappe la classe `Book` à la table `ProductCatalog`, et la propriété `Id` à la clé de partition de clé primaire de la table.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

Dynamo DBIgnore

Indique que la propriété associée devrait être ignorée. Si vous ne souhaitez enregistrer aucune de vos propriétés de classe, vous pouvez ajouter cet attribut afin de demander à `DynamoDBContext` de ne pas inclure cette propriété lors de l'enregistrement d'objets dans la table.

Dynamo DBLocal SecondaryIndexRangeKey

Mappe une propriété de classe à la clé de tri d'un index secondaire local. Utilisez cet attribut si vous devez interroger (`Query`) un index secondaire local, et souhaitez affiner vos résultats à l'aide de la clé de tri d'index.

Dynamo DBProperty

Mappe une propriété de classe à un attribut de table. Si la propriété de classe mappe à un attribut de table du même nom, vous n'avez pas besoin de spécifier cet attribut. Toutefois, si les noms ne sont pas les mêmes, vous pouvez utiliser cette étiquette pour indiquer le mappage. Dans l'instruction C# suivante, la `DynamoDBProperty` mappe la propriété `BookAuthors` à l'attribut `Authors` dans la table.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` utilise ces informations de mappage pour créer l'attribut `Authors` lors de l'enregistrement de données d'objet dans la table correspondante.

Dynamo DBRenamable

Spécifie un autre nom pour une propriété de classe. Ceci est utile si vous écrivez un convertisseur personnalisé pour mapper des données arbitraires à une table DynamoDB où le nom d'une propriété de classe diffère du nom d'un attribut de table.

Clé Dynamo DBRange

Mappe une propriété de classe à la clé de tri de la clé primaire de la table. Si la table possède une clé primaire composite (clé de partition et clé de tri), vous devez spécifier les attributs `DynamoDBHashKey` et `DynamoDBRangeKey` dans votre mappage de classe.

Par exemple, l'exemple de table `Reply` a une clé primaire composée de la clé de partition `Id` et de la clé de tri `Replenishment`. L'exemple de code C# suivant mappe la classe `Reply` à la table `Reply`. La définition de classe indique également que deux de ses propriétés mappent à la clé primaire.

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

Dynamo DBTable

Identifie la table cible dans DynamoDB à laquelle la classe mappe. Par exemple, l'exemple de code C# suivant mappe la classe `Developer` à la table `People` dans DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Cet attribut peut être hérité ou remplacé.

- L'attribut `DynamoDBTable` peut être hérité. Dans l'exemple précédent, si vous ajoutez une nouvelle classe, `Lead`, qui hérite de la classe `Developer`, elle mappe également à la table `People`. Les objets `Developer` et `Lead` sont stockés dans la table `People`.
- L'attribut `DynamoDBTable` peut également être remplacé. Dans l'exemple de code C# suivant, la classe `Manager` hérite de la classe `Developer`. Cependant, l'ajout explicite de l'attribut `DynamoDBTable` a pour effet de mapper la classe à une autre table (`Managers`).

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

Vous pouvez ajouter le paramètre facultatif, `LowerCamelCaseProperties`, pour demander à DynamoDB de mettre la première lettre du nom de propriété en minuscule lors du stockage des objets dans une table, comme dans l'exemple C# suivant.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
```

```
    ...
}
```

Lors de l'enregistrement d'instances de la classe `Developer`, `DynamoDBContext` enregistre la propriété `DeveloperName` en tant que `developerName`.

Dynamo DBVersion

Identifie une propriété de classe pour stocker le numéro de version de l'élément. Pour plus d'informations sur la gestion des versions, consultez Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET.

DBContext Classe Dynamo issue du modèle de persistance des objets .NET

La classe `DynamoDBContext` est le point d'entrée de la base de données Amazon DynamoDB. Elle fournit une connexion à DynamoDB, et vous permet d'accéder à vos données dans diverses tables, d'effectuer diverses opérations CRUD, ainsi que d'exécuter des requêtes. La classe `DynamoDBContext` fournit les méthodes suivantes.

Rubriques

- Créez MultiTable BatchGet
- Créez MultiTable BatchWrite
- CreateBatchGet
- CreateBatchWrite
- Suppression
- Dispose
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Equilibreur de
- Query
- Enregistrer

- Analyser
- ToDocument
- Spécification de paramètres facultatifs pour Dynamo DBContext

Créez MultiTable BatchGet

Crée un objet `MultiTableBatchGet` composé de plusieurs objets `BatchGet`. Chacun de ces objets `BatchGet` peut être utilisé pour récupérer des éléments d'une table DynamoDB.

Pour récupérer les éléments des tables, utilisez la méthode `ExecuteBatchGet` en passant l'objet `MultiTableBatchGet` en tant que paramètre.

Créez MultiTable BatchWrite

Crée un objet `MultiTableBatchWrite` composé de plusieurs objets `BatchWrite`. Chacun de ces objets `BatchWrite` peut être utilisé pour écrire ou supprimer des éléments dans une table DynamoDB.

Pour écrire dans des tables, utilisez la méthode `ExecuteBatchWrite` en passant l'objet `MultiTableBatchWrite` en tant que paramètre.

CreateBatchGet

Crée un objet `BatchGet` que vous pouvez utiliser pour extraire plusieurs éléments d'une table.

CreateBatchWrite

Crée un objet `BatchWrite` que vous pouvez utiliser pour insérer plusieurs éléments dans une table ou supprimer plusieurs éléments d'une table.

Suppression

Supprime un élément de la table. La méthode requiert la clé primaire de l'élément que vous souhaitez supprimer. Vous pouvez fournir en tant que paramètre à cette méthode la valeur de clé primaire ou un objet côté client contenant une valeur de clé primaire.

- Si vous spécifiez un objet côté client en tant que paramètre et avez activé le verrouillage optimiste, la suppression ne réussit que si les versions côté client et côté serveur de l'objet correspondent.
- Si vous spécifiez uniquement la valeur de clé primaire en tant que paramètre, la suppression réussit, que vous ayez activé ou non le verrouillage optimiste.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `DeleteAsync` à la place.

Dispose

Élimine toutes les ressources gérées et non gérées.

ExecuteBatchGet

Lit les données d'une ou plusieurs tables, en traitant tous les objets `BatchGet` en un objet `MultiTableBatchGet`.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `ExecuteBatchGetAsync` à la place.

ExecuteBatchWrite

Ecrit ou supprime des données dans une ou plusieurs tables, en traitant tous les objets `BatchWrite` en un objet `MultiTableBatchWrite`.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `ExecuteBatchWriteAsync` à la place.

FromDocument

Pour une instance de `Document`, la méthode `FromDocument` renvoie une instance d'une classe côté client.

Ceci est utile si vous souhaitez utiliser les classes de modèle de document avec le modèle de persistance des objets pour effectuer des opérations de données. Pour plus d'informations sur les classes de modèles de documents fournies par le AWS SDK pour .NET, consultezUtilisation du modèle de document .NET dans DynamoDB.

Supposons que vous ayez un objet `Document` nommé `doc` qui contient une représentation d'un l'élément `Forum`. (Pour voir comment construire cet objet, consultez la description de la méthode `ToDocument`, plus loin dans cette rubrique. Vous pouvez utiliser le paramètre `FromDocument` pour récupérer l'élément `Forum` à partir de `Document`, comme illustré dans l'exemple de code C# suivant.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Si votre objet `Document` implémente l'interface `IEnumerable`, vous pouvez utiliser la méthode `FromDocuments` à la place. Cela vous permet d'itérer sur toutes les instances de classe dans le `Document`.

FromQuery

Exécute une opération `Query` avec les paramètres de requête définis dans un objet `QueryOperationConfig`.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `FromQueryAsync` à la place.

FromScan

Exécute une opération `Scan` avec les paramètres d'analyse définis dans un objet `ScanOperationConfig`.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `FromScanAsync` à la place.

GetTargetTable

Récupère la table cible pour le type spécifié. Ceci est utile si vous écrivez un convertisseur personnalisé pour mapper des données arbitraires à une table DynamoDB, et devez déterminer la table associée à un type de données personnalisé.

Equilibreur de

Récupère un élément dans une table. La méthode requiert uniquement la clé primaire de l'élément que vous souhaitez extraire.

Par défaut, DynamoDB renvoie l'élément ayant des valeurs éventuellement cohérentes. Pour plus d'informations sur le modèle de cohérence éventuelle, consultez Cohérence en lecture DynamoDB.

`Load`ou la `LoadAsync` méthode appelle l'GetItemopération, qui vous oblige à spécifier la clé primaire de la table. Comme `GetItem` ignore le paramètre `IndexName`, vous ne pouvez pas charger un élément à l'aide de la partition ou de la clé de tri d'un index. Vous devez donc utiliser la clé primaire de la table pour charger un élément.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `LoadAsync` à la place. Pour voir un exemple d'utilisation de la méthode `LoadAsync` pour effectuer des opérations CRUD de haut niveau sur une table DynamoDB, consultez l'exemple suivant.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
```

```
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
```

```
    }
```

Query

Interroge une table sur la base des paramètres de requête que vous fournissez.

Vous pouvez interroger une table uniquement si elle comporte une clé primaire composite (clé de partition et clé de tri). Lors de l'interrogation, vous devez spécifier une clé de partition et une condition qui s'applique à la clé de tri.

Supposons que vous disposez d'une classe `Reply` côté client mappée à la table `Reply` dans DynamoDB. L'exemple de code C# suivant interroge la table `Reply` pour trouver les réponses des unités d'exécution de forum publiées au cours des 15 derniers jours. La table `Reply` possède une clé primaire qui a la clé de partition `Id` et la clé de tri `ReplyDateTime`.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

Ceci renvoie une collection d'objets `Reply`.

La méthode `Query` renvoie une collection `IEnumerable` « avec chargement différé ». Elle ne renvoie initialement qu'une seule page de résultats puis effectue un appel de service pour la page suivante si nécessaire. Pour obtenir tous les éléments correspondants, vous devez itérer uniquement sur la collection `IEnumerable`.

Si votre table possède une clé primaire simple (clé de partition), vous ne pouvez pas utiliser la méthode `Query`. Au lieu de cela, vous pouvez utiliser la méthode `Load` et fournir la clé de partition pour récupérer l'élément.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `QueryAsync` à la place.

Enregistrer

Enregistre l'objet spécifié dans la table. Si la clé primaire spécifiée dans l'objet d'entrée n'existe pas dans la table, la méthode ajoute un nouvel élément à celle-ci. Si la clé primaire existe, la méthode met à jour l'élément existant.

Si le verrouillage optimiste est configuré, la mise à jour ne réussit que si les versions client et serveur de l'élément correspondent. Pour de plus amples informations, veuillez consulter Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `SaveAsync` à la place.

Analyser

Effectue une analyse de la table toute entière.

Vous pouvez filtrer les résultats d'analyse en spécifiant une condition d'analyse. La condition peut être évaluée sur n'importe quel attribut dans la table. Supposons que vous disposez d'une classe `Book` côté client mappée à la table `ProductCatalog` dans DynamoDB. L'exemple C # suivant analyse la table et renvoie uniquement les éléments livre dont le prix est inférieur à 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

La méthode `Scan` renvoie une collection `IEnumerable` « avec chargement différé ». Elle ne renvoie initialement qu'une seule page de résultats puis effectue un appel de service pour la page suivante si nécessaire. Pour obtenir tous les éléments correspondants, vous devez seulement itérer sur la collection `IEnumerable`.

Pour des raisons de performance, vous devez interroger vos tables et éviter une analyse de table.

> **Note**
>
> Pour effectuer cette opération en arrière-plan, utilisez la méthode `ScanAsync` à la place.

ToDocument

Renvoie une instance de la classe de modèle de document `Document` de votre instance de classe.

Ceci est utile si vous souhaitez utiliser les classes de modèle de document avec le modèle de persistance des objets pour effectuer des opérations de données. Pour plus d'informations sur les classes de modèles de documents fournies par le AWS SDK pour .NET, consultezUtilisation du modèle de document .NET dans DynamoDB.

Supposons que vous disposez d'une classe côté client mappée à l'exemple de table `Forum`. Vous pouvez ensuite utiliser un `DynamoDBContext` pour obtenir un élément en tant qu'objet `Document` à partir de la table `Forum`, comme illustré dans l'exemple de code C# suivant.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Spécification de paramètres facultatifs pour Dynamo DBContext

Lors de l'utilisation du modèle de persistance des objets, vous pouvez spécifier les paramètres facultatifs suivants pour le `DynamoDBContext`.

- **`ConsistentRead`** – Lors de la récupération de données à l'aide des opérations `Load`, `Query` ou `Scan`, vous pouvez ajouter ce paramètre facultatif pour demander les dernières valeurs pour les données.
- **`IgnoreNullValues`** – Ce paramètre informe `DynamoDBContext` d'ignorer les valeurs null sur les attributs lors d'une opération `Save`. Si ce paramètre a la valeur false (ou s'il n'est pas défini), une valeur null est interprétée comme une directive pour supprimer l'attribut spécifique.
- **`SkipVersionCheck`** – Ce paramètre informe `DynamoDBContext` ne pas comparer les versions lors de l'enregistrement ou de la suppression d'un élément. Pour plus d'informations sur la gestion

des versions, consultez Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET.

- **`TableNamePrefix`** – Préfixe tous les noms de table avec une chaîne spécifique. Si ce paramètre a la valeur null (ou s'il n'est pas défini), aucun préfixe n'est utilisé.
- `DynamoDBEntryConversion` – Spécifie le schéma de conversion utilisé par le client. Vous pouvez définir ce paramètre sur la version V1 ou V2. V1 est la version par défaut.

  En fonction de la version que vous définissez, le comportement de ce paramètre change. Par exemple :

  - Dans la version 1, le type de données `bool` est converti en type numérique `N`, où 0 représente false et 1 représente true. Dans la version 2, `bool` est converti en `BOOL`.
  - Dans la version 2, les listes et les tableaux ne sont pas regroupés avec HashSets. Les listes et tableaux de nombres, de types basés sur des chaînes et de types binaires sont convertis en type `L` (Liste), qui peut être envoyé vide pour mettre à jour une liste. Ce n'est pas le cas de la version 1, dans laquelle une liste vide n'est pas envoyée sur le réseau.

    Dans la version 1, les types de collection, tels que List HashSet, et les tableaux sont traités de la même manière. La liste et le tableau de chiffres sont convertis au type `NS` (ensemble de nombres). HashSet

  L'exemple suivant définit la version du schéma de conversion sur V2, ce qui modifie le comportement de conversion entre les types .NET et les types de données DynamoDB.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

L'exemple de code C# suivant crée un nouveau `DynamoDBContext` en spécifiant deux des paramètres facultatifs précédents, `ConsistentRead` et `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
```

```
      new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` inclut ces paramètres facultatifs avec chaque demande que vous envoyez à l'aide de ce contexte.

Au lieu de définir ces paramètres au niveau de `DynamoDBContext`, vous pouvez les spécifier pour des opérations individuelles que vous exécutez à l'aide de `DynamoDBContext`, comme illustré dans l'exemple de code C# suivant. L'exemple charge un élément livre spécifique. La méthode `Load` de `DynamoDBContext` spécifie les paramètres facultatifs précédents, `ConsistentRead` et `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

Dans ce cas, `DynamoDBContext` inclut ces paramètres uniquement lors de l'envoi de la demande `Get`.

Verrouillage optimiste utilisant DynamoDB utilisant le modèle et l'objet de persistance des objets AWS SDK pour .NET

La prise en charge du verrouillage optimiste dans le modèle de persistance des objets garantit que la version de l'élément pour votre application est identique à la version de l'élément côté serveur avant de mettre à jour ou de supprimer l'élément. Supposons que vous récupérez un élément à mettre à jour. Toutefois, avant de renvoyer vos mises à jour, une autre application met à jour le même élément. Maintenant, votre demande a une copie obsolète de l'élément. A défaut de verrouillage optimiste, toute mise à jour que vous effectuez remplace la mise à jour effectuée par l'autre application.

La fonction de verrouillage optimiste du modèle de persistance des objets fournit l'étiquette `DynamoDBVersion` que vous pouvez utiliser pour activer le verrouillage optimiste. Pour utiliser cette fonctionnalité, vous ajoutez une propriété à votre classe pour stocker le numéro de version. Vous ajoutez l'attribut `DynamoDBVersion` à la propriété. Lorsque vous enregistrez l'objet pour la première fois, le `DynamoDBContext` attribue un numéro de version et incrémente cette valeur chaque fois que vous mettez à jour l'élément.

Votre demande de mise à jour ou de suppression aboutit uniquement si la version de l'objet côté client correspond au numéro de version de l'élément côté serveur. Si votre application a une copie obsolète de l'élément, elle doit obtenir du serveur la dernière version de l'élément avant de pouvoir le mettre à jour ou le supprimer.

L'exemple de code C# suivant définit une classe `Book` avec des attributs de persistance des objets qui la mappent à la table `ProductCatalog`. La propriété `VersionNumber` dans la classe décorée avec l'attribut `DynamoDBVersion` stocke la valeur du numéro de version.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> **Note**
>
> Vous pouvez appliquer l'attribut `DynamoDBVersion` uniquement à un type primitif numérique nullable (tel que `int?`).

Le verrouillage optimiste a l'impact suivant sur les opérations `DynamoDBContext` :

- Pour un nouvel élément, `DynamoDBContext` attribue le numéro de version initial 0. Si vous récupérez un élément existant, mettez à jour une ou plusieurs de ses propriétés, puis tentez d'enregistrer les modifications, l'opération d'enregistrement n'aboutit que si les numéros de version côté client et côté serveur correspondent. `DynamoDBContext` incrémente le numéro de version. Vous n'avez pas besoin de définir le numéro de version.

- La méthode `Delete` fournit des surcharges pouvant prendre une valeur de clé primaire ou un objet en tant que paramètre, comme illustré dans l'exemple de code C# suivant.

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  // Do other operations.
  // Delete 1 - Pass in the book object.
  context.Delete<ProductCatalog>(book);

  // Delete 2 - Pass in the Id (primary key)
  context.Delete<ProductCatalog>(222);
  ```

  Si vous fournissez un objet en tant que paramètre, la suppression ne réussit que si la version de l'objet correspond à la version de l'élément côté serveur. Toutefois, si vous fournissez une valeur de clé primaire en tant que paramètre, `DynamoDBContext` ne connaît aucun numéro de version, et supprime l'élément sans effectuer la vérification de version.

  Notez que l'implémentation interne du verrouillage optimiste dans le code du modèle de persistance des objets utilise les actions d'API de mise à jour conditionnelle et de suppression conditionnelle dans DynamoDB.

Désactivation du verrouillage optimiste

Pour désactiver le verrouillage optimiste, utilisez la propriété de configuration `SkipVersionCheck`. Vous pouvez définir cette propriété lors de la création de `DynamoDBContext`. Dans ce cas, le verrouillage optimiste est désactivé pour toutes les demandes que vous effectuez à l'aide du contexte. Pour de plus amples informations, veuillez consulter Spécification de paramètres facultatifs pour Dynamo DBContext .

Au lieu de définir la propriété au niveau du contexte, vous pouvez désactiver le verrouillage optimiste pour une opération spécifique, comme dans l'exemple de code C# suivant. L'exemple utilise le contexte pour supprimer un élément livre. La méthode `Delete` définit la propriété `SkipVersionCheck` sur true, ce qui a pour effet de désactiver la vérification de version.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

Mappage de données arbitraires avec DynamoDB à l'aide AWS SDK pour .NET du modèle de persistance des objets

En plus des types .NET pris en charge (voir Types de données pris en charge), vous pouvez utiliser des types dans votre application pour lesquels il n'y a pas de mappage direct aux types Amazon DynamoDB. Le modèle de persistance des objets prend en charge le stockage de données de types arbitraires tant que vous fournissez le convertisseur pour convertir les données du type arbitraire au type DynamoDB, et inversement. Le code de convertisseur transforme les données lors de l'enregistrement et du chargement des objets.

Vous pouvez créer n'importe quel type côté client. Cependant, les données stockées dans les tables sont l'un des types DynamoDB et, lors de la requête et de l'analyse, toutes les comparaisons de données effectuées sont faites par rapport aux données stockées dans DynamoDB.

L'exemple de code C# suivant définit une classe `Book` avec les propriétés `Id`, `Title`, `ISBN` et `Dimension`. La propriété `Dimension` est du `DimensionType` qui décrit les propriétés `Height`, `Width` et `Thickness`. L'exemple de code fournit les méthodes de convertisseur `ToEntry` et `FromEntry` pour convertir les données entre le `DimensionType` et les types de chaîne DynamoDB. Par exemple, lors de l'enregistrement d'une instance `Book`, le convertisseur crée une chaîne de `Dimension` de livre telle que « 8.5x11x.05 ». Lorsque vous récupérez un livre, il convertit la chaîne en instance `DimensionType`.

L'exemple mappe le type `Book` à la table `ProductCatalog`. Il enregistre un exemple d'instance `Book`, le récupère, met à jour ses dimensions et enregistre de nouveau le `Book` mis à jour.

Pour step-by-step obtenir des instructions sur le test de l'exemple suivant, reportez-vous àExemples de code .NET.

Example

```
using System;
```

```
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling
 Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);

                // 3. Update property (book dimensions).
                bookRetrieved.Dimensions.Height += 1;
```

```
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;
            // Update the book.
            context.Save(bookRetrieved);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }
}
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey] //Partition key
    public int Id
    {
        get; set;
    }
    [DynamoDBProperty]
    public string Title
    {
        get; set;
    }
    [DynamoDBProperty]
    public string ISBN
    {
        get; set;
    }
    // Multi-valued (set type) attribute.
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors
    {
        get; set;
    }
    // Arbitrary type, with a converter to map it to DynamoDB type.
    [DynamoDBProperty(typeof(DimensionTypeConverter))]
    public DimensionType Dimensions
    {
        get; set;
    }
}
```

```
    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
            get; set;
        }
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();

            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                              bookDimensions.Length, bookDimensions.Height,
bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
string.IsNullOrEmpty((string)primitive.Value))
                throw new ArgumentOutOfRangeException();
```

```
            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
 StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
}
```

Exécution des exemples de code du Guide du développeur

Les kits SDK AWS permettent une prise en charge étendue d'Amazon DynamoDB dans les langages suivants :

- Java
- JavaScript dans le navigateur
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

Les exemples de code de ce guide du développeur couvrent de façon plus approfondie les opérations DynamoDB à l'aide des langages de programmation suivants :

- Exemples de code Java
- Exemples de code .NET

Avant de commencer cet exercice, vous devez vous créer une compte AWS, obtenir votre clé d'accès et votre clé secrète, et configurer l'AWS Command Line Interface (AWS CLI) sur votre ordinateur. Pour de plus amples informations, consultez Configuration de DynamoDB (service web) .

> **Note**
>
> Si vous utilisez la version téléchargeable de DynamoDB, vous devez utiliser l'AWS CLI afin de créer les tables et les exemples de données. Vous devez également spécifier le paramètre `--endpoint-url` avec chaque commande de l'AWS CLI. Pour de plus amples informations, consultez Définition du point de terminaison local .

Création de tables et chargement de données pour des exemples de code dans DynamoDB

Vous trouverez ci-dessous des informations de base sur la création de tables dans DynamoDB, le chargement d'un exemple de jeu de données, l'interrogation des données et la mise à jour des données.

- Étape 1 : création d'une table dans DynamoDB
- Étape 2 : écrire des données dans une table DynamoDB
- Étape 3 : lire des données à partir d'une table DynamoDB
- Étape 4 : mettre à jour des données dans une table DynamoDB

Exemples de code Java

Rubriques

- Java : définition de vos informations d'identification AWS
- Java : définition de la région AWS et du point de terminaison

Ce Guide du développeur contient des extraits de code Java et des programmes prêts à être exécutés. Vous pouvez trouver ces exemples de code dans les sections suivantes :

- Utilisation d'éléments et d'attributs dans DynamoDB
- Utilisation de tables et de données dans DynamoDB
- Interrogation de tables dans DynamoDB

- Analyse de tables dans DynamoDB
- Amélioration de l'accès aux données avec les index secondaires dans DynamoDB
- Java 1.x : DynamoDBMapper
- Modifier la récupération de données pour DynamoDB Streams

Vous pouvez démarrer rapidement en utilisant Eclipse avec AWS Toolkit for Eclipse. En plus d'un IDE complet, vous obtenez aussi l'AWS SDK pour Java avec les mises à jour automatiques, ainsi que des modèles préconfigurés pour la génération d'applications AWS.

Pour exécuter les exemples de code Java (à l'aide d'Eclipse)

1. Téléchargez et installez l'IDE Eclipse.
2. Téléchargez et installez le kit AWS Toolkit for Eclipse.
3. Démarrez Eclipse et, dans le menu Eclipse, choisissez File (Fichier), New (Nouveau), puis Other (Autre).
4. Dans Sélectionner un assistant, choisissez successivement AWS, AWS Projet Java et Suivant.
5. Dans Créer une Java AWS, procédez comme suit :

   a. Dans Project name (Nom de projet), saisissez un nom pour votre projet.

   b. Dans Select Account, choisissez votre profil d'informations d'identification dans la liste.

      S'il s'agit de votre première utilisation de AWS Toolkit for Eclipse, choisissez Configurer des comptes AWS pour configurer vos informations d'identification AWS.
6. Choisissez Finish pour créer le projet.
7. Dans le menu Eclipse, choisissez File, New, puis Class.
8. Dans Java Class (Classe Java), saisissez un nom pour votre classe dans Name (Nom) (utilisez le même nom que celui de l'exemple de code que vous souhaitez exécuter), puis choisissez Finish (Terminer) pour créer la classe.
9. Copiez l'exemple de code de la page de documentation dans l'éditeur Eclipse.
10. Pour exécuter le code, choisissez Run (Exécuter) dans le menu Eclipse.

Le kit SDK pour Java fournit des clients thread-safe à utiliser avec DynamoDB. En tant que bonne pratique, vos applications doivent créer un seul client et le réutiliser entre les threads.

Pour plus d'informations, consultez le AWS SDK pour Java.

> **Note**
>
> Les exemples de code dans ce guide sont destinés à être utilisés avec la dernière version de AWS SDK pour Java.
> Si vous utilisez l'AWS Toolkit for Eclipse, vous pouvez configurer des mises à jour automatiques pour le kit SDK pour Java. Pour ce faire dans Eclipse, accédez à Preferences (Préférences) et choisissez successivement AWS Toolkit, AWS SDK pour Java, Download new SDKs automatically (Télécharger les nouveaux kits SDK automatiquement).

Java : définition de vos informations d'identification AWS

Le kit (SDK) nécessite que vous fournissiez des informations d'identification AWS à votre application lors de l'exécution. Les exemples de code dans ce guide partent du principe que vous utilisez un fichier d'informations d'identification AWS, comme décrit dans la section Configuration de vos informations d'identification AWS du Guide du développeur AWS SDK pour Java.

Voici un exemple de fichier d'informations d'identification AWS nommé `~/.aws/credentials`, où le caractère tilde (~) représente votre répertoire de base.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java : définition de la région AWS et du point de terminaison

Par défaut, les exemples de code accèdent à DynamoDB dans la région USA Ouest (Oregon). Vous pouvez modifier la région en modifiant les propriétés `AmazonDynamoDB`.

L'exemple de code suivant instancie un nouvel `AmazonDynamoDB`.

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

Vous pouvez utiliser la méthode `withRegion` pour exécuter votre code sur DynamoDB dans n'importe quelle région où il est disponible. Pour obtenir la liste complète, consultez Régions et points de terminaison AWS dans le Référence générale d'Amazon Web Services.

Si vous souhaitez exécuter les exemples de code à l'aide de DynamoDB localement sur votre ordinateur, définissez le point de terminaison comme suit.

Kit SDK AWS V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

Kit SDK AWS V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

Exemples de code .NET

Rubriques

- .NET : définition de vos informations d'identification AWS
- .NET : définition de la région AWS et du point de terminaison

Le guide contient des extraits de code .NET et des programmes prêts à être exécutés. Vous pouvez trouver ces exemples de code dans les sections suivantes :

- Utilisation d'éléments et d'attributs dans DynamoDB
- Utilisation de tables et de données dans DynamoDB
- Interrogation de tables dans DynamoDB
- Analyse de tables dans DynamoDB

- Amélioration de l'accès aux données avec les index secondaires dans DynamoDB
- Utilisation du modèle de document .NET dans DynamoDB
- Utilisation du modèle de persistance des objets .NET et de DynamoDB
- Modifier la récupération de données pour DynamoDB Streams

Vous pouvez démarrer rapidement en utilisant l'AWS SDK pour .NET avec le Toolkit for Visual Studio.

Pour exécuter les exemples de code .NET (à l'aide de Visual Studio)

1. Téléchargez et installez Microsoft Visual Studio.
2. Téléchargez et installez le Toolkit pour Visual Studio.
3. Démarrez Visual Studio 2015. Choisissez File (Fichier), New (Nouveau), puis Project (Projet).
4. Dans Nouveau projet, choisissez Projet vide AWS, puis OK.
5. Dans Informations d'identification AWS, choisissez Utiliser un profil existant, sélectionnez votre profil d'informations d'identification dans la liste, puis choisissez OK.

   S'il s'agit de votre première utilisation du Toolkit for Visual Studio, choisissez Use a new profile (Utiliser un nouveau profil) pour configurer vos informations d'identification AWS.
6. Dans votre projet Visual Studio, sélectionnez l'onglet du code source de votre programme (`Program.cs`). Copiez l'exemple de code de la page de documentation dans l'éditeur Visual Studio, en remplaçant tout autre code que vous voyez dans l'éditeur.
7. Si vous voyez des messages d'erreur tels que The type or namespace name...could not be found (Le nom de type ou d'espace de nom...est introuvable), vous devez installer l'assembly du kit SDK AWS pour DynamoDB comme suit :

   a. Dans l'Explorateur de solutions, ouvrez le menu contextuel (clic droit) de votre projet, puis choisissez Manage NuGet Packages.
   b. Dans le gestionnaire de package NuGet, choisissez Browse.
   c. Dans la zone de recherche, saisissez **`AWSSDK.DynamoDBv2`** et attendez la fin de la recherche.
   d. Choisissez AWSSDK. DynamoDBv2, puis Install (Installer).
   e. Lorsque l'installation est terminée, sélectionnez l'onglet Program.cs pour revenir à votre programme.
8. Pour exécuter le code, choisissez Start (Début) dans la barre d'outils Visual Studio.

L'SDK pour .NET fournit des clients thread-safe à utiliser avec DynamoDB. En tant que bonne pratique, vos applications doivent créer un seul client et le réutiliser entre les threads.

Pour plus d'informations, consultez Kit SDK AWS pour .NET.

> **Note**
>
> Les exemples de code dans ce guide sont destinés à être utilisés avec la dernière version de AWS SDK pour .NET.

.NET : définition de vos informations d'identification AWS

L'SDK pour .NET nécessite que vous fournissiez les informations d'identification AWS à votre application lors de l'exécution. Les exemples de code dans ce guide partent du principe que vous utilisez le kit SDK Store pour gérer votre fichier d'informations d'identification AWS, comme décrit dans Utilisation du magasin SDK dans le Guide du développeur AWS SDK pour .NET.

Le Toolkit for Visual Studio prend en charge plusieurs jeux d'informations d'identification d'un nombre quelconque de comptes. Chaque ensemble est référencé comme profil. Visual Studio ajoute des entrées au fichier `App.config` du projet de façon que votre application puisse trouver les informations d'identification AWS lors de l'exécution.

L'exemple suivant affiche le fichier `App.config` par défaut qui est généré lorsque vous créez un projet à l'aide du Toolkit for Visual Studio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

Lors de l'exécution, le programme utilise le jeu d'informations d'identification `default` d'AWS, comme spécifié par l'entrée `AWSProfileName`. Les informations d'identification AWS proprement dites sont conservées dans le kit SDK Store sous forme chiffrée. Le Toolkit for Visual Studio fournit une interface utilisateur graphique pour gérer vos informations d'identification, le tout dans Visual Studio. Pour plus d'informations, consultez Spécification d'informations d'identification dans le Guide de l'utilisateur AWS Toolkit for Visual Studio.

> **Note**
>
> Par défaut, les exemples de code accèdent à DynamoDB dans la région USA Ouest (Oregon). Vous pouvez modifier la région en modifiant l'entrée `AWSRegion` dans le fichier App.config. Vous pouvez définir `AWSRegion` sur n'importe quelle région dans laquelle DynamoDB est disponible. Pour obtenir la liste complète, consultez Régions et points de terminaison AWS dans le Référence générale d'Amazon Web Services.

.NET : définition de la région AWS et du point de terminaison

Par défaut, les exemples de code accèdent à DynamoDB dans la région USA Ouest (Oregon). Vous pouvez modifier la région en modifiant l'entrée `AWSRegion` dans le fichier `App.config`. Ou alors, vous pouvez modifier la région en modifiant les propriétés `AmazonDynamoDBClient`.

L'exemple de code suivant instancie un nouvel `AmazonDynamoDBClient`. Le client est modifié de telle sorte que le code s'exécute sur DynamoDB dans une autre région.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Pour obtenir la liste complète des régions prises en charge, consultez Régions et points de terminaison AWS dans le Référence générale d'Amazon Web Services.

Si vous souhaitez exécuter les exemples de code à l'aide de DynamoDB localement sur votre ordinateur, définissez le point de terminaison comme suit.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

API de bas niveau de DynamoDB

L'API de bas niveau d'Amazon DynamoDB est l'interface au niveau du protocole pour DynamoDB. À ce niveau, chaque requête HTTPs doit être correctement mise en forme et présenter une signature numérique valide.

Ils créent AWS SDKs des requêtes d'API DynamoDB de bas niveau en votre nom et traitent les réponses de DynamoDB. Vous pouvez ainsi vous concentrer sur la logique de votre application, et non sur les détails de bas niveau. Cependant, vous pouvez continuer à bénéficier d'une connaissance de base du fonctionnement de l'API DynamoDB de bas niveau.

Pour plus d'informations sur l'API DynamoDB de bas niveau, consultez la Référence d'API Amazon DynamoDB.

> **Note**
>
> DynamoDB Streams possède sa propre API de bas niveau, distincte de celle de DynamoDB et entièrement prise en charge par le.AWS SDKs
> Pour de plus amples informations, veuillez consulter Modifier la récupération de données pour DynamoDB Streams. Pour l'API DynamoDB Streams de bas niveau, consultez la Référence d'API Amazon DynamoDB Streams Reference.

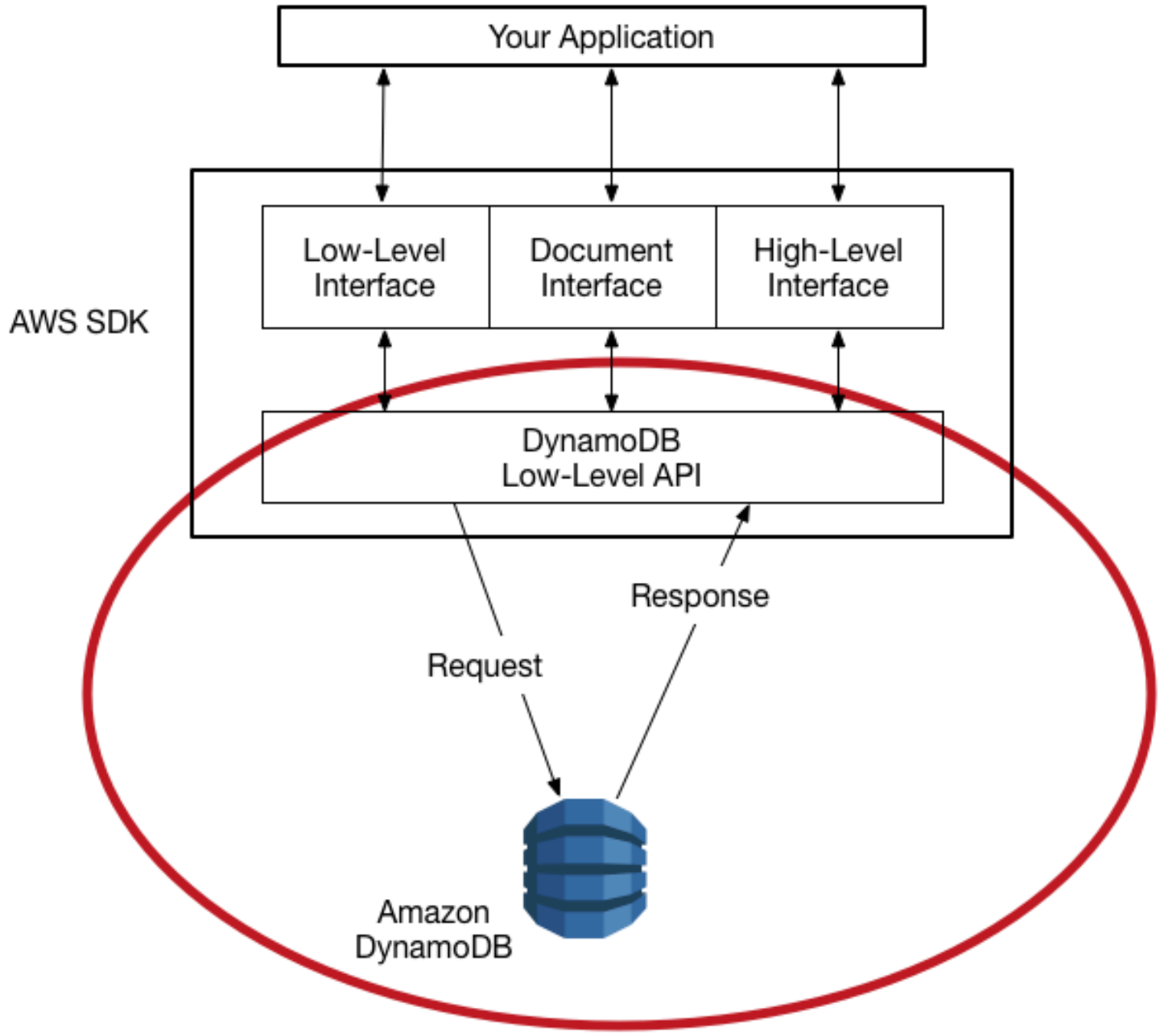
L'API DynamoDB de bas niveau JavaScript utilise la notation d'objet (JSON) comme format de protocole filaire. JSON présentant les données selon une hiérarchie, les valeurs des données et la structure des données sont transmises en même temps. Les paires nom-valeur sont définies au format `name:value`. La hiérarchie des données est définie par les crochets imbriqués des paires nom-valeur.

DynamoDB n'utilise JSON que comme protocole de transport, pas comme format de stockage. Ils AWS SDKs utilisent JSON pour envoyer des données à DynamoDB, et DynamoDB répond par JSON. DynamoDB ne stocke pas de données de manière persistante au format JSON.

> **Note**
>
> Pour plus d'informations sur JSON, consultez Introducing JSON sur le site web `JSON.org`.

Rubriques

- Format des demandes
- Format de la réponse
- Descripteurs de type de données
- Données numériques
- Données binaires



Format des demandes

L'API de bas niveau DynamoDB accepte des requêtes HTTP(S) `POST` en entrée. Ils AWS SDKs élaborent ces demandes pour vous.

Supposons que vous ayez une table nommée `Pets`, avec un schéma de clé composé de `AnimalType` (clé de partition) et `Name` (clé de tri). Ces deux attributs sont de type `string`. Pour récupérer un élément`Pets`, le AWS SDK crée la demande suivante.

```
POST / HTTP/1.1
```

```
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

Notez ce qui suit à propos de cette demande :

- L'en-tête `Authorization` contient les informations requises pour DynamoDB pour authentifier la demande. Pour plus d'informations, voir les demandes d'AWS API de signature et le processus de signature de la version 4 de Signature dans le Référence générale d'Amazon Web Services.
- L'en-tête `X-Amz-Target` contient le nom d'une opération DynamoDB : `GetItem`. (On y trouve aussi la version de l'API de bas niveau, dans ce cas `20120810`.)
- La charge utile (corps) de la demande contient les paramètres de l'opération, au format JSON. Pour l'opération `GetItem`, les paramètres sont `TableName` et `Key`.

Format de la réponse

À la réception de la demande, DynamoDB la traite et renvoie une réponse. Pour la demande illustrée précédemment, la charge utile de la réponse HTTP(S) contient les résultats de l'opération, comme dans l'exemple suivant.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

```
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

À ce stade, le AWS SDK renvoie les données de réponse à votre application pour un traitement ultérieur.

> **Note**
>
> Si DynamoDB ne peut pas traiter une demande, il renvoie un code et un message d'erreur HTTP. Le kit SDK AWS transmet ceux-ci à votre application sous la forme d'exceptions. Pour de plus amples informations, veuillez consulter Gestion des erreurs avec DynamoDB.

Descripteurs de type de données

Le protocole de l'API DynamoDB de bas niveau nécessite que chaque attribut soit accompagné d'un descripteur de type de données. Les descripteurs de type de données sont des jetons qui indiquent à DynamoDB comment interpréter chaque attribut.

Les exemples de Format des demandeset Format de la réponse expliquent comment les descripteurs de type de données sont utilisés. La demande `GetItem` spécifie `S` pour les attributs du schéma de clé de `Pets` (`AnimalType` et `Name`), qui sont de type `string`. La réponse `GetItem` contient un élément Pets avec des attributs de type `string` (`S`), `number` (`N`), `map` (`M`) et `list` (`L`).

La liste suivante est la liste complète des descripteurs de type de données DynamoDB :

- **`S`** – String (chaîne)
- **`N`** – Number (nombre)
- **`B`** – Binary (binaire)
- **`BOOL`** – Boolean (booléen)
- **`NULL`** – Null
- **`M`** – Map (mappage)
- **`L`** – List (liste)
- **`SS`** – String Set (ensemble de chaînes)
- **`NS`** – Number Set (ensemble de nombres)
- **`BS`** – Binary Set (ensemble de binaires)

> **Note**
>
> Pour une description détaillée des types de données DynamoDB, consultez Types de données.

Données numériques

Les différents langages de programmation offrent différents niveaux de prise en charge de JSON. Dans certains cas, vous pouvez décider d'utiliser une bibliothèque tierce pour la validation et l'analyse des documents JSON.

Certaines bibliothèques tierces reposent sur le type de numéro JSON et fournissent leurs propres types, tels que `int`, `long` ou `double`. Toutefois, le type de données Number (nombre) natif dans DynamoDB ne correspondant pas exactement à ces autres types de données, ces distinctions de type peuvent entraîner des conflits. En outre, de nombreuses bibliothèques JSON ne gèrent pas les valeurs numériques de précision fixe et en déduisent automatiquement un type de données double pour les séquences numériques contenant une virgule.

Pour résoudre ces problèmes, DynamoDB propose un seul type numérique sans perte de données. Pour éviter des conversions implicites indésirables en valeur double, DynamoDB utilise des chaînes pour le transfert de données de valeurs numériques. Cette approche offre une flexibilité pour la mise à jour des valeurs d'attribut tout en assurant la sémantique de tri appropriée, comme le placement des valeurs « 01 », « 2 » et « 03 » dans le bon ordre.

Si la précision numérique est importante pour votre application, vous devez convertir les valeurs numériques en chaînes avant de les passer à DynamoDB.

Données binaires

DynamoDB prend en charge les attributs binaires. Cependant, JSON ne prend pas en charge en mode natif le codage binaire. Pour envoyer les données binaires dans une demande, vous devez les encoder au format base64. À la réception de la demande, DynamoDB décode les données en base 64 pour les convertir en binaires.

Le schéma de codage en base 64 que DynamoDB utilise est décrit dans la rubrique RFC 4648 sur le site web de l'Internet Engineering Task Force (IETF).

Programmation d'Amazon DynamoDB avec Python et Boto3

Ce guide de programmation fournit une orientation aux programmeurs qui souhaitent utiliser Amazon DynamoDB avec Python. Découvrez les différentes couches d'abstraction, la gestion de la configuration, la gestion des erreurs, le contrôle des politiques de nouvelles tentatives, la gestion de keep-alive, etc.

Rubriques

- À propos de Boto
- Utilisation de la documentation Boto
- Comprendre les couches d'abstraction du client et des ressources

- Utilisation de la ressource de table batch_writer
- Exemples supplémentaires de code qui explorent les couches client et ressources
- Compréhension de la façon dont les objets Client et Ressource interagissent avec les sessions et les threads
- Personnalisation de l'objet Configuration
- Gestion des erreurs
- Journalisation
- Hooks d'événement
- Pagination et paginateur
- Programmes d'attente

À propos de Boto

Vous pouvez accéder à DynamoDB depuis Python en utilisant le kit AWS SDK officiel pour Python, communément appelé Boto3. Le nom Boto (prononcé boh-toh) vient d'un dauphin d'eau douce originaire du fleuve Amazon. La bibliothèque Boto3 est la troisième version majeure de la bibliothèque, publiée pour la première fois en 2015. La bibliothèque Boto3 est assez volumineuse, car elle prend en charge tous les services AWS, pas seulement DynamoDB. Cette orientation cible uniquement les parties de Boto3 pertinentes pour DynamoDB.

Boto est maintenu et publié par AWS en tant que projet open source hébergé sur GitHub. Il est divisé en deux packages : Botocore et Boto3.

- Botocore fournit les fonctionnalités de bas niveau. Dans Botocore, vous trouverez le client, la session, les informations d'identification, la configuration et les classes d'exception.
- Boto3 s'appuie sur Botocore. Il propose une interface de plus haut niveau, plus pythonique. Plus précisément, il expose une table DynamoDB en tant que ressource et propose une interface plus simple et plus élégante par rapport à l'interface client de niveau inférieur axée sur les services.

Ces projets étant hébergés sur GitHub, vous pouvez consulter le code source, suivre les problèmes en suspens ou soumettre vos propres problèmes.

Utilisation de la documentation Boto

Démarrez avec la documentation Boto grâce aux ressources suivantes :

- Commencez par la section Quickstart qui fournit un point de départ solide pour l'installation du package. Consultez cette page pour obtenir des instructions sur l'installation de Boto3 si ce n'est pas déjà fait (Boto3 est souvent automatiquement disponible dans des services AWS comme AWS Lambda).
- Ensuite, concentrez-vous sur le Guide de DynamoDB de la documentation. Il explique comment effectuer les activités de base de DynamoDB : créer et supprimer une table, manipuler des éléments, exécuter des opérations par lots, exécuter une requête et effectuer une analyse. Ses exemples utilisent l'interface de ressources. Lorsque vous voyez `boto3.resource('dynamodb')`, cela indique que vous utilisez l'interface de ressources de niveau supérieur.
- Après le guide, vous pouvez consulter la Référence de DynamoDB. Cette page d'accueil fournit une liste exhaustive des classes et des méthodes mises à votre disposition. La classe `DynamoDB.Client` s'affiche en haut. Cela fournit un accès de bas niveau à toutes les opérations du plan de contrôle et du plan de données. En bas, vous voyez la classe `DynamoDB.ServiceResource`. Il s'agit de l'interface pythonique de niveau supérieur. Elle vous permet de créer une table, d'effectuer des opérations par lots sur des tables ou d'obtenir une instance `DynamoDB.ServiceResource.Table` pour des actions spécifiques à une table.

Comprendre les couches d'abstraction du client et des ressources

Les deux interfaces que vous allez utiliser sont l'interface client et l'interface de ressources.

- L'interface client de bas niveau fournit un mappage 1-à-1 vers l'API de service sous-jacente. Chaque API proposée par DynamoDB est disponible via le client. Cela signifie que l'interface client peut fournir des fonctionnalités complètes, mais elle est souvent plus détaillée et complexe à utiliser.
- L'interface de ressources de niveau supérieur ne fournit pas de mappage 1-à-1 de l'API de service sous-jacente. Cependant, elle fournit des méthodes qui vous permettent d'accéder plus facilement au service, telles que `batch_writer`.

Voici un exemple d'insertion d'un élément à l'aide de l'interface client. Vous remarquerez que toutes les valeurs sont transmises sous forme de carte avec la clé indiquant leur type (« S » pour chaîne, « N » pour nombre) et leur valeur sous forme de chaîne. C'est ce que l'on appelle le format JSON DynamoDB.

```
import boto3
```

```
dynamodb = boto3.client('dynamodb')

dynamodb.put_item(
    TableName='YourTableName',
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

Voici la même opération `PutItem` à l'aide de l'interface de ressources. La saisie des données est implicite :

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

Si nécessaire, vous pouvez effectuer une conversion entre le JSON normal et le JSON DynamoDB à l'aide des classes `TypeSerializer` et `TypeDeserializer` fournies avec boto3 :

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
```

```
    }

def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

Voici comment effectuer une requête à l'aide de l'interface client. Elle exprime la requête sous forme de construction JSON. Elle utilise une chaîne `KeyConditionExpression` qui nécessite une substitution de variable pour gérer tout conflit de mots clés potentiel :

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

La même opération de requête utilisant l'interface de ressources peut être raccourcie et simplifiée :

```
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
```

```
)
```

Enfin, imaginez que vous souhaitiez obtenir la taille approximative d'une table (c'est-à-dire les métadonnées conservées sur la table et mises à jour toutes les 6 heures environ). Avec l'interface client, vous effectuez une opération `describe_table()` et vous extrayez la réponse de la structure JSON renvoyée :

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Avec l'interface de ressources, la table exécute implicitement l'opération de description et présente les données directement sous forme d'attribut :

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Lorsque vous envisagez de développer à l'aide de l'interface client ou de l'interface de ressources, sachez que les nouvelles fonctionnalités ne seront pas ajoutées à l'interface de ressources conformément à la documentation des ressources : « L'équipe du kit AWS Python SDK n'a pas l'intention d'ajouter de nouvelles fonctionnalités à l'interface de ressources dans boto3. Les interfaces existantes continueront de fonctionner pendant le cycle de vie de boto3. Les clients peuvent accéder aux nouvelles fonctionnalités du service via l'interface client. »

Utilisation de la ressource de table batch_writer

Une commodité disponible uniquement avec la ressource de table de niveau supérieur est `batch_writer`. DynamoDB prend en charge les opérations d'écriture par lots, ce qui permet

d'effectuer jusqu'à 25 opérations de saisie ou de suppression par requête réseau. Un tel traitement par lots améliore l'efficacité en minimisant les allers-retours sur le réseau.

Avec la bibliothèque client de bas niveau, vous utilisez l'opération `client.batch_write_item()` pour exécuter des lots. Vous devez diviser manuellement votre travail en lots de 25. Après chaque opération, vous devez également demander à recevoir une liste des éléments non traités (certaines opérations d'écriture peuvent réussir tandis que d'autres peuvent échouer). Vous devez ensuite retransmettre ces éléments non traités à une opération `batch_write_item()` ultérieure. Il y a une quantité importante de code standard.

La méthode Table.batch_writer crée un gestionnaire de contexte pour écrire des objets dans un lot. Elle présente une interface dans laquelle vous avez l'impression d'écrire des éléments un par un, mais en interne, elle les met en mémoire tampon et les envoie par lots. Elle gère également implicitement les nouvelles tentatives d'éléments non traités.

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

Exemples supplémentaires de code qui explorent les couches client et ressources

Vous pouvez également consulter les référentiels d'exemples de code suivants qui explorent l'utilisation des différentes fonctions, en utilisant à la fois le client et les ressources :

- Exemples de codes à action unique AWS officiels.
- Exemples de code orientés vers des scénarios AWS officiels.
- Exemples de code à action unique gérés par la communauté.

Compréhension de la façon dont les objets Client et Ressource interagissent avec les sessions et les threads

L'objet Resource n'est pas adapté aux threads et ne doit pas être partagé entre les threads ou les processus. Pour plus d'informations, reportez-vous au Guide sur Ressource.

L'objet Client, en revanche, est généralement adapté aux threads, à l'exception de certaines fonctionnalités avancées. Pour plus d'informations, reportez-vous au Guide sur Clients.

L'objet Session n'est pas adapté aux threads. Ainsi, chaque fois que vous créez un objet Client ou Ressource dans un environnement multithread, vous devez d'abord créer un nouvel objet Session, puis créer l'objet Client ou Ressource à partir de Session. Pour plus d'informations, reportez-vous au Guide sur Sessions.

Lorsque vous appelez la `boto3.resource()`, vous utilisez implicitement l'objet Session par défaut. C'est pratique pour écrire du code à thread unique. Lorsque vous écrivez du code multithread, vous devez d'abord créer un nouvel objet Session pour chaque thread, puis récupérer la ressource de cet objet Session :

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Personnalisation de l'objet Configuration

Lorsque vous créez un objet Client ou Ressource, vous pouvez transmettre des paramètres nommés facultatifs pour personnaliser le comportement. Le paramètre nommé `config` déverrouille diverses fonctionnalités. Il s'agit d'une instance de `botocore.client.Config` et la documentation de référence de Configuration montre tout ce qu'il vous expose pour que vous puissiez le contrôler. Le Guide de Configuration fournit une bonne vue d'ensemble.

> **Note**
>
> Vous pouvez modifier bon nombre de ces paramètres comportementaux au niveau de Session, dans le fichier de configuration AWS ou en tant que variables d'environnement.

Configuration pour les délais d'expiration

L'une des utilisations d'une configuration personnalisée consiste à ajuster les comportements réseau :

- connect_timeout (float ou int) : durée en secondes avant qu'une exception de délai d'expiration ne soit déclenchée lors d'une tentative d'établissement d'une connexion. Le durée par défaut est de 60 secondes.
- read_timeout (float ou int) : durée en secondes avant qu'une exception de délai d'expiration ne soit déclenchée lors d'une tentative de lecture à partir d'une connexion. Le durée par défaut est de 60 secondes.

Les délais d'expiration de 60 secondes sont excessifs pour DynamoDB. Cela signifie qu'un problème réseau passager retardera le client d'une minute avant qu'il ne puisse réessayer. Le code suivant réduit les délais d'expiration à une seconde :

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Pour d'autres discussions sur les délais d'expiration, consultez Tuning AWS Java SDK HTTP request settings for latency-aware DynamoDB applications. Notez que le kit Java SDK possède plus de configurations de délai d'expiration que Python.

Configuration pour keep-alive

Si vous utilisez botocore 1.27.84 ou version ultérieure, vous pouvez également contrôler TCP Keep-Alive :

- tcp_keepalive (bool) : active l'option de socket TCP Keep-Alive utilisée lors de la création de nouvelles connexions si elle est définie sur `True` (par défaut sur `False`). Ceci n'est disponible qu'à partir de botocore 1.27.84.

La définition de TCP Keep-Alive sur `True` peut réduire les latences moyennes. Voici un exemple de code qui définit de manière conditionnelle TCP Keep-Alive sur true lorsque vous disposez de la bonne version de botocore :

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion

required_version = "1.27.84"
current_version = botocore.__version__

my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> **Note**
>
> TCP Keep-Alive est différent de HTTP Keep-Alive. Avec TCP Keep-Alive, de petits paquets sont envoyés par le système d'exploitation sous-jacent via la connexion socket afin de maintenir la connexion active et de détecter immédiatement toute perte. Avec HTTP Keep-Alive, la connexion web établie sur le socket sous-jacent est réutilisée. HTTP Keep-Alive est toujours activé avec boto3.

Il y a une limite à la durée pendant laquelle une connexion inactive peut être maintenue en vie. Envisagez d'envoyer des demandes périodiques (disons toutes les minutes) si vous avez une connexion inactive mais que vous souhaitez que la prochaine demande utilise une connexion déjà établie.

Configuration pour les nouvelles tentatives

La configuration accepte également un dictionnaire appelé nouvelles tentatives dans lequel vous pouvez spécifier le comportement de nouvelle tentative souhaité. Les nouvelles tentatives se produisent dans le kit SDK lorsque celui-ci reçoit une erreur et que l'erreur est de type passager. Si une erreur fait l'objet d'une nouvelle tentative en interne (et qu'une nouvelle tentative produit finalement une réponse de réussite), aucune erreur n'est détectée du point de vue du code appelant, juste une latence légèrement élevée. Voici les valeurs que vous pouvez spécifier :

- max_attempts : entier représentant le nombre maximal de nouvelles tentatives qui seront effectuées sur une seule demande. Par exemple, si vous définissez cette valeur sur 2, la demande fait l'objet d'un maximum de deux nouvelles tentatives après la demande initiale. Si cette valeur est définie sur 0, aucune nouvelle tentative n'a lieu après la demande initiale.
- total_max_attempts : entier représentant le nombre maximal de nouvelles tentatives qui seront effectuées sur une seule demande. Cela inclut la demande initiale, donc une valeur de 1 indique qu'aucune demande ne fera l'objet d'une nouvelle tentative. Si `total_max_attempts` et `max_attempts` sont fournis, `total_max_attempts` a priorité. `total_max_attempts` et est préféré à `max_attempts`, car elle correspond à la variable d'environnement `AWS_MAX_ATTEMPTS` et à la valeur de fichier de configuration `max_attempts`.
- mode : chaîne représentant le type de mode de nouvelle tentative que botocore doit utiliser. Les valeurs valides sont :
  - legacy : mode par défaut. Attend 50 ms lors de la première nouvelle tentative, puis utilise un backoff exponentiel avec un facteur de base de 2. Pour DynamoDB, il effectue jusqu'à 10 tentatives maximum au total (sauf si vous remplacez la valeur par la valeur ci-dessus).

    > **Note**
    >
    > Avec un backoff exponentiel, la dernière tentative attendra près de 13 secondes.

  - standard : nommé « standard », car il est plus cohérent avec les autres kits AWS SDK. Attend pendant un temps aléatoire compris entre 0 ms et 1 000 ms pour la première nouvelle tentative. Si une autre nouvelle tentative est nécessaire, il choisit un autre temps aléatoire compris entre 0 ms et 1 000 ms et le multiplie par 2. Si une nouvelle tentative supplémentaire est nécessaire, il effectue le même choix aléatoire multiplié par 4, et ainsi de suite. Chaque attente est limitée à 20 secondes. Ce mode effectue de nouvelles tentatives sur un plus grand nombre de conditions de défaillance détectées que le mode `legacy`. Pour DynamoDB, il effectue jusqu'à 3 tentatives maximum au total (sauf si vous remplacez la valeur par la valeur ci-dessus).
  - adaptative : mode de nouvelle tentative expérimental qui inclut toutes les fonctionnalités du mode standard, mais ajoute une limitation automatique côté client. Grâce à la limitation adaptative du débit, les kits SDK peuvent ralentir le rythme d'envoi des demandes afin de mieux tenir compte de la capacité des services AWS. Il s'agit d'un mode provisoire dont le comportement est susceptible de changer.

Vous trouverez une définition détaillée de ces modes de nouvelle tentative dans le Guide des nouvelles tentatives ainsi que dans Comportement des tentatives dans la référence du kit SDK.

Voici un exemple qui utilise explicitement la politique de nouvelles tentatives `legacy` avec un maximum de 3 demandes au total (2 nouvelles tentatives) :

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
   retries = {
     'mode': 'legacy',
     'total_max_attempts': 3
   }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

DynamoDB étant un système à haute disponibilité et à faible latence, vous souhaiterez peut-être adopter une vitesse de nouvelle tentative plus rapide que ne le permettent les politiques de nouvelles tentatives intégrées. Vous pouvez implémenter votre propre politique de nouvelles tentatives en fixant le nombre maximum de tentatives à 0, en détectant vous-même les exceptions et en réessayant le cas échéant à partir de votre propre code au lieu de compter sur boto3 pour effectuer des nouvelles tentatives implicites.

Si vous gérez votre propre stratégie de nouvelle tentative, vous devez faire la différence entre les limitations et les erreurs :

- Une limitation (indiquée par une `ProvisionedThroughputExceededException` ou une `ThrottlingException`) indique un service en bonne santé qui vous informe que vous avez dépassé votre capacité de lecture ou d'écriture sur une table ou une partition DynamoDB. Chaque milliseconde qui passe, un peu plus de capacité de lecture ou d'écriture est disponible, ce qui vous permet d'effectuer rapidement une nouvelle tentative (par exemple toutes les 50 ms) pour tenter d'accéder à cette capacité nouvellement libérée. Avec les limitations, vous n'avez pas particulièrement besoin de backoff exponentiel, car les limitations sont légères pour que DynamoDB les renvoie et n'entraînent aucun frais par demande pour vous. Le backoff exponentiel affecte des délais plus longs aux threads client qui ont déjà attendu le plus longtemps, ce qui étend statistiquement les p50 et p99 vers l'extérieur.
- Une erreur (indiquée par `InternalServerError` ou `ServiceUnavailable`, entre autres) indique un problème passager avec le service. Cela peut concerner l'ensemble de la table ou simplement la partition sur laquelle vous lisez ou sur laquelle vous écrivez. En cas d'erreur,

vous pouvez faire une pause plus longue avant de d'effectuer de nouvelles tentatives (250 ms ou 500 ms, par exemple) et utiliser l'instabilité pour échelonner les nouvelles tentatives.

Configuration pour le nombre maximum de connexions au groupe

Enfin, la configuration vous permet de contrôler la taille du groupe de connexions :

- max_pool_connections (int) : nombre maximum de connexions à conserver dans un groupe de connexions. Si cette valeur n'est pas spécifiée, la valeur par défaut de 10 est utilisée.

Cette option contrôle le nombre maximum de connexions HTTP à conserver en groupe en vue de leur réutilisation. Un groupe différent est conservé par session. Si vous prévoyez que plus de 10 threads vont à l'encontre de clients ou de ressources créés à partir de la même session, vous devriez envisager de l'augmenter, afin que les threads n'aient pas à attendre d'autres threads utilisant une connexion groupée.

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)

# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

Gestion des erreurs

Les exceptions de service AWS ne sont pas toutes définies de manière statique dans Boto3. Cela est dû au fait que les erreurs et les exceptions liées aux services AWS varient considérablement et sont sujettes à changement. Boto3 enveloppe toutes les exceptions de service sous forme de `ClientError` et expose les détails en tant que JSON structuré. Par exemple, une réponse d'erreur peut être structurée comme suit :

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
    }
}
```

Le code suivant détecte toutes les exceptions `ClientError` et examine la valeur de chaîne de caractères du `Code` dans `Error` pour déterminer l'action à entreprendre :

```
import botocore
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
        print("Received a server error")
    else:
        raise err
```

Certains codes d'exception (mais pas tous) ont été matérialisés sous forme de classes de premier niveau. Vous pouvez choisir de les gérer directement. Lorsque vous utilisez l'interface Client, ces

exceptions sont renseignées dynamiquement sur le client et vous pouvez les intercepter à l'aide de l'instance client, comme suit :

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Lorsque vous utilisez l'interface Ressource, vous devez utiliser `.meta.client` pour passer de la ressource au Client sous-jacent pour accéder aux exceptions, comme ceci :

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Pour consulter la liste des types d'exception matérialisés, vous pouvez générer la liste de manière dynamique :

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Lorsque vous effectuez une opération d'écriture avec une expression de condition, vous pouvez demander qu'en cas d'échec de l'expression, la valeur de l'élément soit renvoyée dans la réponse d'erreur.

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

Pour en savoir plus sur la gestion des erreurs et les exceptions, consultez :

- Le guide boto3 sur la gestion des erreurs contient plus d'informations sur les techniques de gestion des erreurs.
- La section du guide du développeur DynamoDB consacrée aux erreurs de programmation répertorie les erreurs que vous pouvez rencontrer.
- La section Erreurs courantes de la référence de l'API.
- La documentation de chaque opération d'API répertorie les erreurs que cet appel peut générer (par exemple BatchWriteItem).

Journalisation

La bibliothèque boto3 s'intègre au module de journalisation intégré de Python pour suivre ce qui se passe pendant une session. Pour contrôler les niveaux de journalisation, vous pouvez configurer le module de journalisation :

```
import logging

logging.basicConfig(level=logging.INFO)
```

Cela configure l'enregistreur racine pour qu'il journalise `INFO` et les messages de niveau supérieur. Les messages de journalisation moins graves que le niveau sont ignorés. Les niveaux de journalisation peuvent être `DEBUG`, `INFO`, `WARNING`, `ERROR` et `CRITICAL`. L'argument par défaut est `WARNING`.

Les enregistreurs de boto3 sont hiérarchiques. La bibliothèque utilise plusieurs enregistreurs différents, chacun correspondant à différentes parties de la bibliothèque. Vous pouvez contrôler séparément le comportement de chacun :

- boto3 : enregistreur principal du module boto3.
- botocore : enregistreur principal du package botocore.
- botocore.auth : utilisé pour enregistrer la création de signatures AWS pour les demandes.
- botocore.credentials : utilisé pour journaliser le processus de récupération et d'actualisation des informations d'identification.
- botocore.endpoint : utilisé pour journaliser la création de demandes avant leur envoi sur le réseau.
- botocore.hooks : utilisé pour journaliser les événements déclenchés dans la bibliothèque.
- botocore.loaders : utilisé pour la journalisation lorsque des parties de modèles de service AWS sont chargées.
- botocore.parsers : utilisé pour journaliser les réponses du service AWS avant qu'elles ne soient analysées.
- botocore.retryhandler : utilisé pour journaliser le traitement des nouvelles tentatives de demande de service AWS (mode hérité).
- botocore.retries.standard : utilisé pour journaliser le traitement des nouvelles tentatives de demande de service AWS (mode standard ou adaptatif).
- botocore.utils : utilisé pour journaliser diverses activités dans la bibliothèque.

- botocore.waiter : utilisé pour enregistrer les fonctionnalités des programmes d'attente, qui interrogent un service AWS jusqu'à ce qu'un certain état soit atteint.

D'autres bibliothèques utilisent également les journaux. En interne, boto3 utilise le fichier tiers urllib3 pour la gestion des connexions HTTP. Lorsque la latence est importante, vous pouvez consulter ses journaux pour vous assurer que votre pool est bien utilisé en voyant quand urllib3 établit une nouvelle connexion ou ferme une connexion inactive.

- urllib3.connectionpool : à utiliser pour journaliser les événements de gestion du groupe de connexions.

L'extrait de code suivant définit la plupart des connexions sur la journalisation dans `INFO` avec la journalisation `DEBUG` pour l'activité du point de terminaison et du groupe de connexions :

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

Hooks d'événement

Botocore émet des événements au cours des différentes étapes de son exécution. Vous pouvez enregistrer des gestionnaires pour ces événements afin que chaque fois qu'un événement est émis, votre gestionnaire soit appelé. Cela vous permet d'étendre le comportement du botocore sans avoir à modifier les composants internes.

Supposons, par exemple, que vous souhaitiez suivre chaque fois qu'une opération `PutItem` est appelée sur une table DynamoDB de votre application. Vous pouvez vous enregistrer sur l'événement `'provide-client-params.dynamodb.PutItem'` à capturer et journaliser chaque fois qu'une opération `PutItem` est invoquée dans la session associée. Voici un exemple :

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
    if 'TableName' in params and 'Item' in params:
```

```
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

Dans le gestionnaire, vous pouvez même manipuler les paramètres par programmation pour modifier le comportement :

```
params['TableName'] = "NewTableName"
```

Pour plus d'informations sur les événements, consultez botocore documentation on events et boto3 documentation on events.

Pagination et paginateur

Certaines demandes, telles que Query et Scan, limitent la taille des données renvoyées lors d'une seule demande et nécessitent que vous fassiez des demandes répétées pour extraire les pages suivantes.

Vous pouvez contrôler le nombre maximum d'éléments à lire pour chaque page à l'aide du paramètre `limit`. Par exemple, si vous voulez extraire uniquement les 10 derniers éléments, vous pouvez utiliser le paramètre `limit`. Notez que la limite correspond à la quantité qui doit être lue dans la table avant qu'un filtrage ne soit appliqué. Il n'est pas possible de spécifier que vous voulez

exactement 10 éléments après le filtrage ; vous ne pouvez contrôler le nombre préfiltré et vérifier côté client que lorsque vous en avez réellement extrait 10. Quelle que soit la limite, chaque réponse a toujours une taille maximale de 1 Mo.

Si la réponse inclut une `LastEvaluatedKey`, cela indique qu'elle s'est terminée parce qu'elle a atteint une limite de nombre ou de taille. La clé est la dernière clé évaluée pour la réponse. Vous pouvez extraire cette `LastEvaluatedKey` et la transmettre à un appel de suivi en tant que `ExclusiveStartKey` afin de lire le fragment suivant à partir de ce point de départ. Si aucune `LastEvaluatedKey` n'est renvoyée, cela signifie qu'il n'y a plus d'éléments correspondant à Query ou Scan.

Voici un exemple simple (utilisant l'interface Ressource, mais l'interface Client suit le même schéma) qui lit au maximum 100 éléments par page et boucle jusqu'à ce que tous les éléments aient été lus.

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}

while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

Pour plus de commodité, boto3 peut le faire pour vous avec des paginateurs. Cependant, cela ne fonctionne qu'avec l'interface Client. Voici le code réécrit pour utiliser les paginateurs :

```
import boto3
```

```
dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')

query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

Pour plus d'informations, consultez le Guide on Paginators et l'API reference for DynamoDB.Paginator.Query.

> **Note**
>
> Les paginateurs ont également leurs propres paramètres de configuration nommés `MaxItems`, `StartingToken` et `PageSize`. Pour la pagination avec DynamoDB, vous devez ignorer ces paramètres.

Programmes d'attente

Les programmes d'attente offrent la possibilité d'attendre que quelque chose soit terminé avant de continuer. À l'heure actuelle, ils ne prennent en charge que l'attente de la création ou de la suppression d'une table. En arrière-plan, le programme d'attente effectue une vérification pour vous toutes les 20 secondes, jusqu'à 25 fois. Vous pouvez le faire vous-même mais l'utilisation d'un programme d'attente est élégante lors de l'automatisation de l'écriture.

Ce code montre comment attendre qu'une table particulière ait été créée :

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

Pour plus d'informations, consultez le Guide to Waiters et la Reference on Waiters.

Programmation d'Amazon DynamoDB avec JavaScript

Ce guide fournit une orientation aux programmeurs qui souhaitent utiliser Amazon DynamoDB avec JavaScript. Découvrez AWS SDK pour JavaScript, les couches d'abstraction disponibles, la configuration de connexions, le traitement des erreurs, la définition de politiques de nouvelle tentative, la gestion de keep-alive, et bien d'autres fonctionnalités.

Rubriques

- À propos de AWS SDK pour JavaScript
- Utilisation de la version 3 de AWS SDK pour JavaScript
- Accès à la documentation JavaScript
- Couches d'abstraction
- Utilisation de la fonction utilitaire de regroupement
- Lecture d'éléments
- Écritures conditionnelles
- Pagination
- Spécification d'une configuration
- Programmes d'attente
- Gestion des erreurs
- Journalisation
- Considérations

À propos de AWS SDK pour JavaScript

AWS SDK pour JavaScript permet d'accéder à des Services AWS en utilisant des scripts de navigateur ou Node.js. Cette documentation se concentre sur la dernière version du kit SDK (V3). La version 3 de AWS SDK pour JavaScript est gérée par AWS e, tant que projet open source hébergé

sur GitHub. Les problèmes et les demandes de fonctionnalités sont publics et vous pouvez y accéder sur la page des problèmes du référentiel GitHub.

La version 2 de JavaScript est similaire à la version 3, mais elle contient des différences de syntaxe. La version 3 est plus modulaire, ce qui facilite l'expédition de petites dépendances, et dispose d'une prise en charge TypeScript de première classe. Nous vous recommandons d'utiliser la dernière version du kit SDK.

Utilisation de la version 3 de AWS SDK pour JavaScript

Vous pouvez ajouter le kit SDK à votre application Node.js à l'aide de Node Package Manager. Les exemples ci-dessous montrent comment ajouter les packages SDK les plus courants à utiliser avec DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

L'installation de packages ajoute des références à la section de dépendance de votre fichier de projet package.json. Vous avez la possibilité d'utiliser la nouvelle syntaxe du module ECMAScript. Pour plus de détails sur ces deux approches, consultez la section Considérations.

Accès à la documentation JavaScript

Démarrez avec la documentation JavaScript grâce aux ressources suivantes :

- Accédez au guide du développeur pour la documentation de base de JavaScript. Les instructions d'installation se trouvent dans la section Configuration.
- Accédez à la documentation de référence de l'API pour explorer toutes les classes et méthodes disponibles.
- Le kit SDK pour JavaScript prend en charge de nombreux autres Services AWS que DynamoDB. Utilisez la procédure suivante pour localiser la couverture d'API spécifique pour DynamoDB :

  1. Dans Services, sélectionnez DynamoDB and Libraries. Il s'agit de la documentation pour le client de bas niveau.
  2. Choisissez lib-dynamodb. Il s'agit de la documentation pour le client de haut niveau. Les deux clients représentent deux couches d'abstraction différentes que vous avez le choix d'utiliser. Consultez la section ci-dessous pour plus d'informations sur les couches d'abstraction.

Couches d'abstraction

Le kit SDK pour JavaScript version V3 possède un client de bas niveau (`DynamoDBClient`) et un client de haut niveau (`DynamoDBDocumentClient`).

Rubriques

- Client de bas niveau (DynamoDBClient)
- Client de haut niveau (DynamoDBDocumentClient)

Client de bas niveau (**`DynamoDBClient`**)

Le client de bas niveau ne fournit aucune abstraction supplémentaire par rapport au protocole filaire sous-jacent. Cela vous donne un contrôle total sur tous les aspects de la communication, mais comme il n'y a pas d'abstractions, vous devez notamment fournir des définitions d'éléments à l'aide du format JSON DynamoDB.

Comme le montre l'exemple ci-dessous, avec ce format, les types de données doivent être indiqués explicitement. Un S indique une valeur de chaîne et un N une valeur numérique. Les numéros sur la transmission sont toujours envoyés sous forme de chaînes balisées sous forme de types numériques pour éviter toute perte de précision. Les appels d'API de bas niveau ont un modèle d'affectation tel que `PutItemCommand` et `GetItemCommand`.

L'exemple suivant utilise un client de bas niveau `Item` défini à l'aide d'un fichier JSON DynamoDB :

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };
```

```
  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Client de haut niveau (**`DynamoDBDocumentClient`**)

Le client de document DynamoDB de haut niveau offre des fonctionnalités pratiques intégrées, telles que l'élimination de la nécessité de rassembler manuellement les données et l'autorisation de lectures et d'écritures directes à l'aide d'objets JavaScript standard. La documentation de `lib-dynamodb` fournit la liste des avantages.

Pour instancier le `DynamoDBDocumentClient`, construisez un `DynamoDBClient` de bas niveau ; puis encapsulez-le avec un `DynamoDBDocumentClient`. La convention de dénomination des fonctions diffère légèrement entre les deux packages. Par exemple, le bas niveau utilise `PutItemCommand` tandis que le haut niveau utilise `PutCommand`. Les noms distincts permettent aux deux ensembles de fonctions de coexister dans le même contexte, ce qui vous permet de combiner les deux dans le même script.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };
```

```
  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

Le modèle d'utilisation est cohérent lorsque vous lisez des éléments à l'aide d'opérations d'API telles que `GetItem`, `Query` ou `Scan`.

Utilisation de la fonction utilitaire de regroupement

Vous pouvez utiliser le client de bas niveau et regrouper ou désorganiser vous-même les types de données. Le package d'utilitaires, util-dynamodb, possède une fonction utilitaire `marshall()` qui accepte le JSON et produit DynamoDB JSON, ainsi qu'une fonction `unmarshall()` qui effectue l'inverse. L'exemple suivant utilise le client de bas niveau avec la désorganisation des données gérée par l'appel `marshall()`.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
```

```
  }
}
addProduct();
```

Lecture d'éléments

Pour lire un seul élément à partir de DynamoDB, utilisez l'opération d'API `GetItem`. Comme pour la commande `PutItem`, vous avez le choix d'utiliser le client de bas niveau ou le client Document de haut niveau. L'exemple ci-dessous montre comment utiliser le client Document de haut niveau pour extraire un élément.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };

  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Utilisez l'opération d'API `Query` pour lire plusieurs éléments. Vous pouvez utiliser le client de bas niveau ou le client Document. L'exemple ci-dessous utilise le client Document de haut niveau.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
      "#category": "category",
      "#sku": "sku",
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

productSearch();
```

Écritures conditionnelles

Les opérations d'écriture DynamoDB peuvent spécifier une expression de condition logique qui doit être définie sur true pour que l'écriture puisse se poursuivre. Si la condition n'est pas définie sur true, l'opération d'écriture génère une exception. L'expression de condition peut vérifier si l'élément existe déjà ou si ses attributs correspondent à certaines contraintes.

`ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"`

Lorsque l'expression conditionnelle échoue, vous pouvez utiliser `ReturnValuesOnConditionCheckFailure` pour demander que la réponse d'erreur inclue

l'élément qui ne satisfaisait pas aux conditions afin de déduire l'origine du problème. Pour plus de détails, consultez Handle conditional write errors in high concurrency scenarios with Amazon DynamoDB.

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
      if (e.name === 'ConditionalCheckFailedException') {
          console.log('Item already exists:', e.Item);
      } else {
          throw e;
      }
  }
```

Des exemples de code supplémentaires illustrant d'autres aspects de l'utilisation du kit SDK JavaScript version 3 sont disponibles dans la documentation du kit SDK JavaScript version 3 et dans le référentiel GitHub DynamoDB-SDK-Examples.

Pagination

Rubriques

- Utilisation de la méthode pratique paginateScan

Les demandes de lecture telles que `Scan` ou `Query` renverront probablement plusieurs éléments dans un jeu de données. Si vous effectuez une `Scan` ou une `Query` avec un paramètre `Limit`, une fois que le système aura lu autant d'éléments, une réponse partielle sera envoyée et vous devrez paginer pour récupérer des éléments supplémentaires.

Le système ne lira qu'un maximum de 1 mégaoctet de données par demande. Si vous incluez une expression `Filter`, le système lira toujours un mégaoctet, au maximum, de données sur le disque, mais renverra les éléments de ce mégaoctet qui correspondent au filtre. L'opération de filtrage peut renvoyer 0 élément pour une page, mais nécessite tout de même une pagination supplémentaire avant que la recherche ne soit épuisée.

Vous devez rechercher `LastEvaluatedKey` dans la réponse et l'utiliser comme paramètre `ExclusiveStartKey` dans une demande ultérieure pour poursuivre l'extraction des données. Cela sert de signet, comme indiqué dans l'exemple suivant.

> **Note**
>
> L'exemple transmet une valeur nulle `lastEvaluatedKey` comme `ExclusiveStartKey` lors de la première itération, ce qui est autorisé.

Exemple d'utilisation de `LastEvaluatedKey` :

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;

  do {
    const params = {
      TableName: "products",
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}

paginatedScan().catch((err) => {
  console.error(err);
});
```

Utilisation de la méthode pratique **`paginateScan`**

Le kit SDK fournit des méthodes pratiques appelées `paginateScan` et `paginateQuery` qui font cela à votre place et qui font les demandes répétées en arrière-plan. Spécifiez le nombre maximum d'éléments à lire par demande à l'aide du paramètre standard `Limit`.

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };

  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}

paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
});
```

> **Note**
>
> Il n'est pas recommandé d'effectuer régulièrement des analyses complètes de la table, sauf si celle-ci est petite.

Spécification d'une configuration

Rubriques

- Configuration pour les délais d'expiration
- Configuration pour keep-alive
- Configuration pour les nouvelles tentatives

Lors de la configuration de `DynamoDBClient`, vous pouvez spécifier différentes remplacements de configuration en transmettant un objet de configuration au constructeur. Par exemple, vous pouvez spécifier la région à laquelle vous connecter si le contexte d'appel ne la connaît pas déjà ou si l'URL du point de terminaison à utiliser ne la connaît pas déjà. Cela est utile si vous souhaitez cibler une instance DynamoDB local à des fins de développement.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Configuration pour les délais d'expiration

DynamoDB utilise HTTPS pour les communications client-serveur. Vous pouvez contrôler certains aspects de la couche HTTP en fournissant un objet `NodeHttpHandler`. Par exemple, vous pouvez ajuster les principales valeurs de délai d'expiration `connectionTimeout` et `requestTimeout`. `connectionTimeout` représente la durée maximale, en millisecondes, pendant laquelle le client doit attendre lorsqu'il essaie d'établir une connexion avant d'abandonner.

`requestTimeout` définit le temps pendant lequel le client doit attendre une réponse après l'envoi d'une demande, également en millisecondes. La valeur par défaut pour les deux est zéro, ce qui signifie que le délai d'attente est désactivé et qu'il n'y a aucune limite quant au temps d'attente du client si la réponse n'arrive pas. Vous devez définir les délais d'expiration à un niveau raisonnable afin qu'en cas de problème réseau, la demande soit en erreur et qu'une nouvelle demande puisse être initiée. Par exemple :

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});

const client = new DynamoDBClient({
  requestHandler
});
```

> **Note**
>
> L'exemple fourni utilise l'importation Smithy. Smithy est un langage de définition de services et de kit SDK, open source et géré par AWS.

Outre la configuration des valeurs de délai d'expiration, vous pouvez définir le nombre maximum de sockets, ce qui permet d'augmenter le nombre de connexions simultanées par origine. Le guide du développeur contient des informations détaillées sur la configuration du paramètre `maxSockets`.

Configuration pour keep-alive

Lorsque vous utilisez le protocole HTTPS, la première demande nécessite toujours des échanges de va-et-vient pour établir une connexion sécurisée. HTTP Keep-Alive permet aux requêtes suivantes de réutiliser la connexion déjà établie, ce qui les rend plus efficaces et réduit le temps de latence. HTTP Keep-Alive est activé par défaut avec la version 3 de JavaScript.

Il y a une limite à la durée pendant laquelle une connexion inactive peut être maintenue en vie. Envisagez d'envoyer des demandes périodiques, par exemple toutes les minutes, si vous avez une connexion inactive mais que vous souhaitez que la prochaine demande utilise une connexion déjà établie.

> **Note**
>
> Notez que dans l'ancienne version 2 du kit SDK, keep-alive était désactivé par défaut, ce qui signifie que chaque connexion était fermée immédiatement après utilisation. Si vous utilisez la version 2, vous pouvez annuler ce paramètre.

Configuration pour les nouvelles tentatives

Lorsque le kit SDK reçoit une réponse d'erreur et que l'erreur peut être reprise comme indiqué par le kit SDK, par exemple une exception de limitation ou une exception de service temporaire, il réessaie. Cela se produit de manière invisible pour vous en tant qu'appelant, sauf que vous remarquerez peut-être que la demande a mis plus de temps à aboutir.

Le kit SDK pour JavaScript version 3 effectuera 3 demandes au total, par défaut, avant d'abandonner et de transmettre l'erreur au contexte d'appel. Vous pouvez ajuster le nombre et la fréquence de ces nouvelles tentatives.

Le constructeur `DynamoDBClient` accepte un paramètre `maxAttempts` qui limite le nombre de tentatives possibles. L'exemple ci-dessous augmente la valeur de 3 par défaut à 5 au total. Si vous le définissez sur 0 ou 1, cela indique que vous ne souhaitez pas de nouvelles tentatives automatiques et que vous souhaitez gérer vous-même les erreurs reprenant dans votre bloc catch.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

Vous pouvez également contrôler le calendrier des nouvelles tentatives à l'aide d'une stratégie de nouvelle tentative personnalisée. Pour ce faire, importez le package d'utilitaires `util-retry` et créez une fonction de sauvegarde personnalisée qui calcule le temps d'attente entre les tentatives en fonction du nombre de tentatives actuel.

L'exemple ci-dessous indique d'effectuer un maximum de 5 tentatives avec des délais de 15, 30, 90 et 360 millisecondes en cas d'échec de la première tentative. La fonction d'annulation personnalisée, `calculateRetryBackoff`, calcule les délais en acceptant le nombre de nouvelles tentatives (commence par 1 pour la première nouvelle tentative) et renvoie le nombre de millisecondes à attendre pour cette demande.

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");

const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

Programmes d'attente

Le client DynamoDB inclut deux fonctions de programmes d'attente qui peuvent être utilisées lors de la création, de la modification ou de la suppression de tables lorsque vous souhaitez que votre code attende jusqu'à ce que la modification des tables soit terminée. Par exemple, vous pouvez déployer une table, appeler la fonction `waitUntilTableExists`, et le code bloquera jusqu'à ce

que la table soit ACTIVE. Le serveur interroge en interne le service DynamoDB avec un `describe-table` toutes les 20 secondes.

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

La fonctionnalité `waitUntilTableExists` ne rend le contrôle que lorsqu'elle peut exécuter une commande `describe-table` indiquant le statut de table ACTIF. Cela garantit que vous pouvez utiliser `waitUntilTableExists` pour attendre la fin de la création, ainsi que des modifications telles que l'ajout d'un index GSI, dont l'application peut prendre un certain temps avant que la table ne revienne au statut ACTIF.

Gestion des erreurs

Dans les premiers exemples présentés ici, nous avons identifié toutes les erreurs de manière générale. Cependant, dans les applications pratiques, il est important de distinguer les différents types d'erreurs et de mettre en œuvre une gestion plus précise de celles-ci.

Les réponses aux erreurs DynamoDB contiennent des métadonnées, notamment le nom de l'erreur. Vous pouvez détecter les erreurs puis les comparer aux noms de chaîne possibles des conditions d'erreur afin de déterminer la marche à suivre. Pour les erreurs côté serveur, vous pouvez optimiser l'opérateur `instanceof` avec les types d'erreur exportés par le package `@aws-sdk/client-dynamodb` pour gérer efficacement la gestion des erreurs.

Il est important de noter que ces erreurs ne se manifestent qu'une fois toutes les nouvelles tentatives épuisées. Si une erreur fait l'objet d'une nouvelle tentative, suivi d'un appel qui aboutit, du point de vue du code, aucune erreur n'est détectée, juste une latence légèrement élevée. Les nouvelles tentatives apparaîtront dans les graphiques Amazon CloudWatch en tant que demandes infructueuses, telles que les demandes de limitation ou d'erreur. Si le client atteint le nombre maximal de nouvelles tentatives, il abandonne et génère une exception. Il s'agit de la façon dont le client dit qu'il ne va pas réessayer.

Vous trouverez ci-dessous un extrait permettant de détecter l'erreur et de prendre des mesures en fonction du type d'erreur renvoyé.

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
      } else {
        // Handle other errors.
      }
    }
}
```

Pour plus d'informations sur les chaînes d'erreur les plus courantes, consultez the section called "Gestion des erreurs" dans le guide du développeur DynamoDB. Les erreurs exactes possibles avec un appel d'API particulier peuvent être trouvées dans la documentation de cet appel d'API, telle que la documentation de l'API de requête.

Les métadonnées des erreurs incluent des propriétés supplémentaires, en fonction de l'erreur. Pour un `TimeoutError`, les métadonnées incluent le nombre de tentatives effectuées et le `totalRetryDelay`, comme indiqué ci-dessous.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
```

```
  }
}
```

Si vous gérez votre propre stratégie de nouvelle tentative, vous devez faire la différence entre les limitations et les erreurs :

- Une limitation (indiquée par une `ProvisionedThroughputExceededException` ou une `ThrottlingException`) indique un service en bonne santé qui vous informe que vous avez dépassé votre capacité de lecture ou d'écriture sur une table ou une partition DynamoDB. Chaque milliseconde qui passe, un peu plus de capacité de lecture ou d'écriture est disponible, ce qui vous permet d'effectuer rapidement une nouvelle tentative (par exemple toutes les 50 ms) pour tenter d'accéder à cette capacité nouvellement libérée.

  Avec les limitations, vous n'avez pas particulièrement besoin de backoff exponentiel, car les limitations sont légères pour que DynamoDB les renvoie et n'entraînent aucun frais par demande pour vous. Le backoff exponentiel affecte des délais plus longs aux threads client qui ont déjà attendu le plus longtemps, ce qui étend statistiquement les p50 et p99 vers l'extérieur.
- Une erreur (indiquée par `InternalServerError` ou `ServiceUnavailable`, entre autres) indique un problème transitoire avec le service, peut-être avec la table entière ou simplement avec la partition sur laquelle vous lisez ou sur laquelle vous écrivez. En cas d'erreur, vous pouvez faire une pause plus longue avant de d'effectuer de nouvelles tentatives (250 ms ou 500 ms, par exemple) et utiliser l'instabilité pour échelonner les nouvelles tentatives.

Journalisation

Activez la journalisation pour obtenir plus de détails sur le fonctionnement du kit SDK. Vous pouvez définir un paramètre sur le`DynamoDBClient`, comme indiqué dans l'exemple ci-dessous. D'autres informations de journal apparaîtront dans la console et incluent des métadonnées telles que le code d'état et la capacité consommée. Si vous exécutez le code localement dans une fenêtre de terminal, les journaux y apparaissent. Si vous exécutez le code dans AWS Lambda et que les journaux Amazon CloudWatch sont configurés, la sortie de la console y sera écrite.

```
const client = new DynamoDBClient({
  logger: console
});
```

Vous pouvez également vous connecter aux activités internes du kit SDK et effectuer une journalisation personnalisée lorsque certains événements se produisent. L'exemple ci-dessous utilise

le `middlewareStack` du client pour intercepter chaque demande au fur et à mesure qu'elle est envoyée par le kit SDK et l'enregistre au fur et à mesure qu'elle se produit.

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
    step: "build",
    name: "log-ddb-calls",
  }
);
```

Le `MiddlewareStack` constitue un hook puissant pour observer et contrôler le comportement du kit SDK. Consultez le blog Introducing Middleware Stack in Modular AWS SDK pour JavaScript pour plus d'informations.

Considérations

Lorsque vous implémentez l'AWS SDK pour JavaScript dans votre projet, voici d'autres facteurs à prendre en compte.

Systèmes de modules

Le kit SDK prend en charge deux systèmes de modules, CommonJS et ES (ECMAScript). CommonJS utilise la fonction `require`, tandis que ES utilise le mot clé `import`.

1. Common JS – `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript – `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

Le type de projet dicte le système de modules à utiliser et est spécifié dans la section type de votre fichier package.json. Le type par défaut est CommonJS. Utilisez `"type": "module"` pour indiquer un projet ES. Si vous avez un projet Node.JS existant qui utilise le format de package CommonJS, vous pouvez toujours ajouter des fonctions avec la syntaxe d'importation plus moderne du kit SDK V3 en nommant vos fichiers de fonctions avec l'extension .mjs. Cela permettra au fichier de code d'être traité comme ES (ECMAScript).

Opérations asynchrones

Vous verrez de nombreux exemples de code utilisant des rappels et des promesses pour gérer le résultat des opérations DynamoDB. Avec le JavaScript moderne, cette complexité n'est plus nécessaire et les développeurs peuvent tirer parti de la syntaxe async/await plus succincte et plus lisible pour les opérations asynchrones.

Exécution du navigateur Web

Les développeurs Web et mobiles qui créent avec React ou React Native peuvent utiliser le kit SDK pour JavaScript dans leurs projets. Avec la version V2 antérieure du kit SDK, les développeurs Web devaient charger le kit SDK complet dans le navigateur, en faisant référence à une image du kit SDK hébergée sur https://sdk.amazonaws.com/js/.

Avec la V3, il est possible de regrouper uniquement les modules clients V3 requis et toutes les fonctions JavaScript requises dans un seul fichier JavaScript à l'aide de Webpack, et de l'ajouter dans une balise de script dans le `<head>` de vos pages HTML, comme expliqué dans la section Démarrage dans un script de navigateur de la documentation du kit SDK.

Opérations de plan de données DAX

Les opérations du plan de données Amazon DynamoDB Streams Accelerator (DAX) sont prises en charge par le kit SDK pour JavaScript V3.

Programmation de DynamoDB avec AWS SDK for Java 2.x

Ce guide de programmation fournit une orientation aux programmeurs qui souhaitent utiliser Amazon DynamoDB avec Java. Le guide couvre différents concepts, notamment les couches d'abstraction, la gestion de la configuration, la gestion des erreurs, le contrôle des politiques de nouvelles tentatives et la gestion de keep-alive.

Rubriques

- À propos de AWS SDK for Java 2.x
- Mise en route avec l' AWS SDK for Java 2.x
- Consultation de la documentation du kit AWS SDK for Java 2.x
- Interfaces prises en charge
- Exemples supplémentaires de code
- Programmation synchrone et asynchrone

- Clients HTTP
- Configuration d'un client HTTP
- Gestion des erreurs
- AWS ID de demande
- Logging
- Pagination
- Annotations de classes de données

À propos de AWS SDK for Java 2.x

Vous pouvez accéder à DynamoDB depuis Java en utilisant le logiciel officiel. AWS SDK pour Java Le kit SDK pour Java est disponible en deux versions : 1.x et 2.x. Le end-of-support for 1.x a été annoncé le 12 janvier 2024. Il entrera en mode maintenance le 31 juillet 2024 et sa date d'échéance end-of-support est fixée au 31 décembre 2025. Pour les nouveaux développements, nous vous recommandons vivement d'utiliser la version 2.x, qui a été publiée pour la première fois en 2018. Ce guide cible exclusivement la version 2.x et se concentre uniquement sur les parties du kit SDK pertinentes pour DynamoDB.

Pour plus d'informations sur la maintenance et le support du AWS SDKs, consultez la politique de maintenance du AWS SDK et des outils et la matrice de prise en charge des versions des outils dans le guide de référence des outils AWS SDKs et des outils.AWS SDKs

AWS SDK for Java 2.x Il s'agit d'une réécriture majeure de la base de code 1.x. Le SDK pour Java 2.x prend en charge les fonctionnalités Java modernes, telles que le système I/O non bloquant introduit dans Java 8. Le kit SDK pour Java 2.x prend également en charge les implémentations de clients HTTP enfichables afin de fournir une plus grande flexibilité de connexion réseau et des options de configuration supplémentaires.

Un changement notable entre le kit SDK pour Java 1.x et le kit SDK pour Java 2.x est l'utilisation d'un nouveau nom de package. Le kit Java 1.x SDK utilise le nom du package `com.amazonaws`, tandis que le kit Java 2.x SDK utilise `software.amazon.awssdk`. De même, les artefacts Maven du kit SDK Java 1.x utilisent le `groupId` `com.amazonaws`, tandis que les artefacts du kit SDK Java 2.x utilisent le `groupId` `software.amazon.awssdk`.

> **Important**
>
> La version AWS SDK pour Java 1.x possède un package DynamoDB nommé. `com.amazonaws.dynamodbv2` Le « v2 » dans le nom du package n'indique pas qu'il s'agit de Java 2 (J2SE). « v2 » indique plutôt que le package prend en charge la seconde version de l'API de bas niveau DynamoDB au lieu de la version d'origine de l'API de bas niveau.

Support pour les versions de Java

AWS SDK for Java 2.x Il fournit un support complet pour les versions Java de support à long terme (LTS).

Mise en route avec l' AWS SDK for Java 2.x

Le didacticiel suivant montre comment utiliser Apache Maven pour définir des dépendances pour le kit SDK pour Java 2.x. Ce didacticiel explique également comment écrire le code qui se connecte à DynamoDB pour répertorier les tables DynamoDB disponibles. Le didacticiel de ce guide est basé sur le didacticiel Démarrer avec le kit AWS SDK for Java 2.x du Guide du développeur du kit AWS SDK for Java 2.x . Nous avons modifié ce didacticiel pour appeler DynamoDB au lieu d'Amazon S3.

Pour terminer ce didacticiel, procédez comme suit :

- Étape 1 : Configuration pour ce didacticiel
- Étape 2 : Création du projet
- Étape 3 : Écriture du code
- Étape 4 : Création et exécution de l'application

Étape 1 : Configuration pour ce didacticiel

Avant de commencer ce didacticiel, vous aurez besoin des éléments suivants :

- Autorisation d'accès à DynamoDB.
- Un environnement de développement Java configuré avec un accès par authentification unique à Services AWS l'aide du Portail d'accès AWS.

Pour procéder à la configuration pour ce didacticiel, suivez les instructions de Présentation de la configuration dans le Guide du développeur du kit AWS SDK for Java 2.x . Une fois que vous avez

configuré votre environnement de développement avec un accès par authentification unique pour le kit Java SDK et que vous disposez d'une session de portail d'accès AWS active, passez à l'étape 2 de ce didacticiel.

Étape 2 : Création du projet

Pour créer le projet pour ce didacticiel, vous devez exécuter une commande Maven qui vous invite à saisir des informations sur la configuration du projet. Une fois toutes les entrées saisies et confirmées, Maven termine la création du projet en créant un fichier `pom.xml` et des fichiers Java stub.

1. Ouvrez un terminal ou une fenêtre d'invite de commande et naviguez jusqu'au répertoire de votre choix, par exemple, le dossier `Desktop` ou `Home`.
2. Saisissez la commande suivante sur le terminal et appuyez sur Entrée.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Pour chaque invite, saisissez la valeur indiquée dans la deuxième colonne.

Invite	Valeur à saisir
`Define value for property 'service':`	`dynamodb`
`Define value for property 'httpClient' :`	`apache-client`
`Define value for property 'nativeImage' :`	`false`
`Define value for property 'credentialProvider'`	`identity-center`
`Define value for property 'groupId':`	`org.example`

Invite	Valeur à saisir
`Define value for property 'artifactId':`	`getstarted`
`Define value for property 'version' 1.0-SNAPSHOT:`	*<Enter>*
`Define value for property 'package' org.example:`	*<Enter>*

4. Une fois que vous avez saisi la dernière valeur, Maven répertorie les choix que vous avez effectués. Pour confirmer, saisissez O. Vous pouvez également saisir N, puis saisir à nouveau vos choix.

Maven crée un dossier de projet nommé `getstarted` en fonction de la valeur `artifactId` que vous avez saisie. Dans le dossier `getstarted`, recherchez un fichier nommé `README.md` que vous pouvez consulter, un fichier `pom.xml` et un répertoire `src`.

Maven crée l'arborescence de répertoires suivante.

```
getstarted
 ### README.md
 ### pom.xml
 ### src
     ### main
     #   ### java
     #   #   ### org
     #   #       ### example
     #   #           ### App.java
     #   #           ### DependencyFactory.java
     #   #           ### Handler.java
     #   ### resources
     #       ### simplelogger.properties
     ### test
         ### java
             ### org
                 ### example
                     ### HandlerTest.java

 10 directories, 7 files
```

L'exemple suivant affiche le contenu du fichier du projet `pom.xml`.

`pom.xml`

La section `dependencyManagement` contient une dépendance vis-à-vis du kit AWS SDK for Java 2.x, et la section `dependencies` possède une dépendance pour DynamoDB. La spécification de ces dépendances oblige Maven à inclure les fichiers `.jar` appropriés dans le chemin de la classe Java. Par défaut, le AWS SDK n'inclut pas toutes les classes pour tous Services AWS. Pour DynamoDB, si vous utilisez l'interface de bas niveau, vous devez avoir une dépendance à l'égard de l'artefact `dynamodb`. Ou, si vous utilisez l'interface de haut niveau, sur l'artefact `dynamodb-enhanced`. Si vous n'incluez pas les dépendances pertinentes, le code ne peut pas être compilé. Le projet utilise Java 1.8 en raison de la valeur `1.8` des propriétés `maven.compiler.source` et `maven.compiler.target`.

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.example</groupId>
     <artifactId>getstarted</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>jar</packaging>
     <properties>
         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
         <maven.compiler.source>1.8</maven.compiler.source>
         <maven.compiler.target>1.8</maven.compiler.target>
         <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
         <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
         <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
         <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <-------- SDK version
 picked up from archetype version.
         <slf4j.version>1.7.28</slf4j.version>
         <junit5.version>5.8.1</junit5.version>
     </properties>

     <dependencyManagement>
         <dependencies>
             <dependency>
                 <groupId>software.amazon.awssdk</groupId>
                 <artifactId>bom</artifactId>
```

```
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>dynamodb</artifactId>  <-------- DynamoDB dependency
            <exclusions>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>netty-nio-client</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sso</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <-------- HTTP client specified.
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
```

```
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

Étape 3 : Écriture du code

Le code suivant présente la classe App créée par Maven. La méthode `main` est le point d'entrée dans l'application, qui crée une instance de la classe `Handler` puis appelle sa méthode `sendRequest`.

classe **App**

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
 }
```

La classe `DependencyFactory` créée par Maven contient la méthode d'usine `dynamoDbClient` qui construit et renvoie une instance `DynamoDbClient`. L'instance `DynamoDbClient` utilise une instance du client HTTP basé sur Apache. Cela est dû au fait que vous avez spécifié `apache-client` quand Maven vous a invité à spécifier le client HTTP à utiliser.

Le code suivant présente la classe `DependencyFactory`.

classe DependencyFactory

```
package org.example;

 import software.amazon.awssdk.http.apache.ApacheHttpClient;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
/**
 * The module containing all dependencies required by the {@link Handler}.
 */
public class DependencyFactory {

    private DependencyFactory() {}

    /**
     * @return an instance of DynamoDbClient
     */
    public static DynamoDbClient dynamoDbClient() {
        return DynamoDbClient.builder()
                       .httpClientBuilder(ApacheHttpClient.builder())
                       .build();
    }
}
```

La classe `Handler` contient la logique principale de votre programme. Lorsqu'une instance de `Handler` est créée dans la classe App, la `DependencyFactory` fournit le client de service `DynamoDbClient`. Votre code utilise l'instance de `DynamoDbClient` pour appeler DynamoDB.

Maven génère la classe `Handler` suivante avec un commentaire *`TODO`*. L'étape suivante du didacticiel consiste à remplacer le commentaire *`TODO`* par du code.

Classe **`Handler`**, générée par Maven

```
package org.example;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        // TODO: invoking the API calls using dynamoDbClient.
    }
}
```

Pour renseigner la logique, remplacez l'ensemble du contenu de la classe `Handler` par le code suivant. La méthode `sendRequest` est renseignée et les importations nécessaires sont ajoutées.

Classe **`Handler`**, implémentée

Le code suivant utilise l'instance `DynamoDbClient` pour extraire la liste des tables existantes. Si des tables existent pour un compte et une Région AWS donnés, le code utilise l'instance `Logger` pour enregistrer les noms de ces tables.

```
package org.example;

 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
 import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
     }

     public void sendRequest() {
         Logger logger = LoggerFactory.getLogger(Handler.class);

         logger.info("calling the DynamoDB API to get a list of existing tables");
         ListTablesResponse response = dynamoDbClient.listTables();

         if (!response.hasTableNames()) {
             logger.info("No existing tables found for the configured account &
 region");
         } else {
             response.tableNames().forEach(tableName -> logger.info("Table: " +
 tableName));
         }
     }
 }
```

Étape 4 : Création et exécution de l'application

Une fois que vous avez créé le projet et qu'il contient la classe `Handler` complète, créez et exécutez l'application.

1. Assurez-vous d'avoir une AWS IAM Identity Center session active. Pour confirmer, exécutez la commande de l' AWS Command Line Interface (AWS CLI) `aws sts get-caller-identity` et vérifiez la réponse. Si vous n'avez pas de session active, consultez Connexion à l'aide de l' AWS CLI pour obtenir des instructions.
2. Ouvrez un terminal ou une fenêtre d'invite de commande et accédez au répertoire de votre projet `getstarted`.
3. Pour créer votre projet, utilisez la commande suivante :

```
mvn clean package
```

4. Pour exécuter l'application, exécutez la commande suivante :

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

Après avoir consulté le fichier, supprimez l'objet, puis le compartiment.

Réussite

Si votre projet Maven a été créé et exécuté sans erreur, félicitations ! Vous avez créé avec succès votre première application Java à l'aide du kit SDK pour Java 2.x.

Nettoyage

Pour nettoyer les ressources que vous avez créées au cours de ce didacticiel, supprimez le dossier du projet`getstarted`.

Consultation de la documentation du kit AWS SDK for Java 2.x

Le Guide du développeur du kit AWS SDK for Java 2.x couvre tous les aspects du kit SDK dans tous les Services AWS. Nous vous recommandons de consulter les rubriques suivantes :

- Migration de la version 1.x vers la version 2.x : inclut une explication détaillée des différences entre les versions 1.x et 2.x. Cette rubrique contient également des instructions sur l'utilisation des deux versions principales side-by-side.

- Guide DynamoDB pour le kit Java 2.x SDK : explique comment effectuer des opérations DynamoDB de base : créer une table, manipuler des éléments et extraire des éléments. Ces exemples utilisent l'interface de bas niveau. Java possède plusieurs interfaces, comme expliqué dans la section suivante : Interfaces prises en charge.

> **Tip**
>
> Après avoir consulté ces rubriques, ajoutez la Référence d'API du kit AWS SDK for Java 2.x à vos favoris. Il couvre tout Services AWS, et nous vous recommandons de l'utiliser comme référence d'API principale.

Interfaces prises en charge

Il AWS SDK for Java 2.x prend en charge les interfaces suivantes, en fonction du niveau d'abstraction souhaité.

Rubriques de cette section

- Interface de bas niveau
- Interface de haut niveau
- Interface de document
- Comparaison d'interfaces avec un exemple de Query

Interface de bas niveau

L'interface de bas niveau fournit un one-to-one mappage vers l'API de service sous-jacente. Chaque API DynamoDB est disponible via cette interface. Cela signifie que l'interface de bas niveau peut fournir des fonctionnalités complètes, mais elle est souvent plus détaillée et complexe à utiliser. Par exemple, vous devez utiliser les fonctions `.s()` pour contenir des chaînes et les fonctions `.n()` pour contenir des nombres. L'exemple suivant montre PutItemcomment insérer un élément à l'aide de l'interface de bas niveau.

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;
```

```
import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

Interface de haut niveau

L'interface de haut niveau du AWS SDK for Java 2.x est appelée client amélioré DynamoDB. Cette interface fournit une expérience de création de code plus idiomatique.

Le client amélioré permet de mapper les classes de données côté client et les tables DynamoDB conçues pour stocker ces données. Vous définissez les relations entre des tables et leurs classes de modèle correspondantes dans votre code. Vous pouvez ensuite compter sur le kit SDK pour gérer la manipulation des types de données. Pour plus d'informations sur le client amélioré, consultez DynamoDB enhanced client API dans le Guide du développeur du kit AWS SDK for Java 2.x .

L'exemple suivant PutItemutilise l'interface de haut niveau. Dans cet exemple, le `DynamoDbBean` nommé `YourItem` crée un `TableSchema` qui permet son utilisation directe comme entrée pour l'appel de `putItem()`.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
```

```
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
            this.inventory = inventory;
        }

        public int getInventory() {
            return inventory;
        }
    }
}
```

La AWS SDK pour Java version 1.x possède sa propre interface de haut niveau, souvent désignée par sa classe `DynamoDBMapper` principale. AWS SDK for Java 2.x Il est publié dans un package distinct (et un artefact Maven) nommé. `software.amazon.awssdk.enhanced.dynamodb` Le kit Java 2.x SDK est souvent désigné par sa classe principale `DynamoDbEnhancedClient`.

Interface de haut niveau utilisant des classes de données immuables

La fonctionnalité de mappage de l'API client améliorée DynamoDB fonctionne également avec des classes de données immuables. Une classe immuable ne possède que des méthodes getter et

nécessite une classe de générateur que le kit SDK utilise pour créer des instances de la classe. L'immuabilité en Java est un style couramment utilisé que les développeurs peuvent utiliser pour créer des classes sans effets secondaires. Le comportement de ces classes est plus prévisible dans les applications multithread complexes. Au lieu d'utiliser l'annotation `@DynamoDbBean` comme indiqué dans l'High-level interface example, les classes immuables utilisent l'annotation `@DynamoDbImmutable`, qui prend la classe de générateur comme entrée.

L'exemple suivant utilise la classe de générateur `DynamoDbEnhancedClientImmutablePutItem` comme entrée pour créer un schéma de table. L'exemple fournit ensuite le schéma comme entrée pour l'appel d'PutItemAPI.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                    .pk("123")
                                    .sk("cart#123")
                                    .itemData("YourItemData")
                                    .inventory(500)
                                    .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

L'exemple suivant illustre la classe de données immuables.

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }

    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public String getSk() {
        return sk;
    }

    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
```

```
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

Interface de haut niveau utilisant des classes de données immuables et des bibliothèques de génération standard tierces

Les classes de données immuables (illustrées dans l'exemple précédent) nécessitent un code standard. Par exemple, la logique getter et setter appliquée aux classes de données, en plus des classes `Builder`. Les bibliothèques tierces, telles que Project Lombok, peuvent vous aider à générer ce type de code standard. La réduction de la majeure partie du code standard vous permet de limiter la quantité de code nécessaire pour travailler avec les classes de données immuables et le SDK. AWS Cela se traduit également par une amélioration de la productivité et de la lisibilité du code. Pour plus d'informations, consultez Use third-party libraries, such as Lombok, dans le Guide du développeur du kit AWS SDK for Java 2.x .

L'exemple suivant montre comment Project Lombok simplifie le code nécessaire à l'utilisation de l'API client améliorée DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
```

```
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

L'exemple suivant illustre l'objet de données immuables de la classe de données immuables.

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
    String pk;
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

La `YourImmutableLombokItem` classe utilise les annotations suivantes fournies par Project Lombok et le AWS SDK :

- @Builder — Produit un générateur complexe APIs pour les classes de données fourni par Project Lombok.
- @ DynamoDbImmutable — Identifie la `DynamoDbImmutable` classe en tant qu'annotation d'entité mappable DynamoDB fournie par le SDK. AWS

- @Value : variante immuable de `@Data`. Par défaut, tous les champs sont définis comme privés et définitifs, et les méthodes setter ne sont pas générées. Project Lombok fournit cette annotation.

Interface de document

L'interface AWS SDK for Java 2.x Document évite d'avoir à spécifier des descripteurs de type de données. Les types de données découlent de la sémantique des données proprement dites. Cette interface de document est similaire à l'interface de document AWS SDK pour Java 1.x, mais avec une interface repensée.

L'Document interface example suivant montre l'appel de `PutItem` exprimé à l'aide de l'interface de document. L'exemple utilise également EnhancedDocument. Pour exécuter des commandes sur une table DynamoDB à l'aide de l'API de document améliorée, vous devez d'abord associer la table au schéma de votre table de documents pour créer un objet de ressource `DynamoDBTable`. Le générateur de schéma de table de documents nécessite les principaux fournisseurs de clés d'index et de convertisseurs d'attributs.

Vous pouvez utiliser `AttributeConverterProvider.defaultProvider()` pour convertir les attributs de document des types par défaut. Vous pouvez modifier le comportement général par défaut à l'aide d'une implémentation de `AttributeConverterProvider` personnalisée. Vous pouvez également modifier le convertisseur pour un seul attribut. Le guide de référence AWS SDKs and Tools fournit plus de détails et des exemples sur l'utilisation de convertisseurs personnalisés. Ils sont principalement utilisés pour les attributs des classes de domaine pour lesquels aucun convertisseur par défaut n'est disponible. À l'aide d'un convertisseur personnalisé, vous pouvez fournir au kit SDK les informations nécessaires pour écrire ou lire dans DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
```

```
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
                                .item(
                                    EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                            .putString("pk", "123")
                                            .putString("sk", "cart#123")
                                            .putString("item_data", "YourItemData")
                                            .putNumber("inventory", 500)
                                            .build())
                                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

}
```

Pour convertir des documents JSON en types de données Amazon DynamoDB natifs, vous pouvez utiliser les méthodes utilitaires suivantes :

- `EnhancedDocument.fromJson(String json)`— Crée une nouvelle EnhancedDocument instance à partir d'une chaîne JSON.
- `EnhancedDocument.toJson()` : crée une représentation sous forme de chaîne JSON du document que vous pouvez utiliser dans votre application comme n’importe quel autre objet JSON.

Comparaison d'interfaces avec un exemple de **`Query`**

Cette section montre le même appel de `Query` exprimé à l'aide des différentes interfaces. Pour optimiser les résultats de ces requêtes, notez ce qui suit :

- DynamoDB cible une valeur de clé de partition spécifique. Vous devez donc spécifier la clé de partition dans son intégralité.
- Pour que la requête cible uniquement les articles du panier, la clé de tri possède une expression de condition clé qui utilise `begins_with`.
- Nous utilisons `limit()` pour limiter la requête à un maximum de 100 articles renvoyés.
- Nous avons défini `scanIndexForward` sur false. Les résultats sont renvoyés dans l'ordre des octets UTF-8, ce qui signifie généralement que l'article du panier avec le numéro le plus bas est renvoyé en premier. En définissant `scanIndexForward` sur false, nous annulons la commande et l'article du panier avec le numéro le plus élevé est renvoyé en premier.
- Nous appliquons un filtre pour supprimer tout résultat ne correspondant pas aux critères. Les données filtrées consomment de la capacité de lecture, que l'élément corresponde au filtre ou non.

Example **`Query`** avec l'interface de bas niveau

L'exemple suivant interroge un élément à partir de la table nommée `YourTableName` à l'aide d'une `keyConditionExpression`. Cela limite la requête à une valeur de clé de partition spécifique et à une valeur de clé de tri commençant par une valeur de préfixe spécifique. Ces conditions clés limitent la quantité de données lues depuis DynamoDB. Enfin, la requête applique un filtre sur les données extraites de DynamoDB à l'aide d'une `filterExpression`.

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT =
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);
```

```
    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

Example **`Query`** avec l'interface de document

L'exemple suivant interroge une table nommée `YourTableName` à l'aide de l'interface de document.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
```

```
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example **`Query`** avec l'interface de haut niveau

L'exemple suivant interroge une table nommée `YourTableName` à l'aide de l'API client améliorée DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
        LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String name) {
            this.pk = pk;
            this.sk = sk;
```

```
            this.name = name;
        }

        private String pk;
        private String sk;
        private String name;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

Interface de haut niveau utilisant des classes de données immuables

Lorsque vous effectuez une opération `Query` avec les classes de données immuables de haut niveau, le code est le même que celui de l'exemple d'interface de haut niveau, à l'exception de la construction de la classe d'entité `YourItem` ou `YourImmutableItem`. Pour plus d'informations, consultez l'PutItemexemple.

Interface de haut niveau utilisant des classes de données immuables et des bibliothèques de génération standard tierces

Lorsque vous effectuez une opération `Query` avec les classes de données immuables de haut niveau, le code est le même que celui de l'exemple d'interface de haut niveau, à l'exception de la construction de la classe d'entité `YourItem` ou `YourImmutableLombokItem`. Pour plus d'informations, consultez l'PutItemexemple.

Exemples supplémentaires de code

Pour des exemples supplémentaires d'utilisation de DynamoDB avec le kit SDK pour Java 2.x, reportez-vous aux référentiels d'exemples de code suivants :

- Exemples de codes à action unique AWS officiels
- Exemples de code à action unique gérés par la communauté
- Exemples de code officiels AWS orientés vers des scénarios

Programmation synchrone et asynchrone

AWS SDK for Java 2.x fournit à la fois des clients synchrones et asynchrones pour Services AWS, tels que DynamoDB.

Les classes `DynamoDbClient` et `DynamoDbEnhancedClient` fournissent des méthodes synchrones qui bloquent l'exécution du thread jusqu'à ce que le client reçoive une réponse du service. Ce client est le moyen le plus simple d'interagir avec DynamoDB si vous n'avez pas besoin d'opérations asynchrones.

Les classes `DynamoDbAsyncClient` et `DynamoDbEnhancedAsyncClient` fournissent des méthodes asynchrones qui renvoient immédiatement, en rendant le contrôle au thread appelant sans attendre de réponse. Le client non bloquant présente l'avantage d'une simultanéité élevée sur quelques threads, ce qui permet de traiter efficacement les demandes d'E/S avec un minimum de ressources de calcul. Cela améliore le débit et la réactivité.

Il AWS SDK for Java 2.x utilise le support natif pour les E/S non bloquantes. La version AWS SDK pour Java 1.x devait simuler des E/S non bloquantes.

Dans la mesure où une méthode synchrone renvoie avant qu'une réponse ne soit disponible, vous avez besoin d'une solution pour obtenir la réponse quand elle est prête. Les méthodes asynchrones AWS SDK pour Java renvoient un `CompletableFuture`objet contenant les résultats

de l'opération asynchrone à venir. Lorsque vous appelez `get()` ou utilisez `join()` sur ces objets `CompletableFuture`, le code se bloque jusqu'à ce que le résultat soit disponible. Si vous les appelez en même temps que vous faites la demande, le comportement est similaire à celui d'un simple appel synchrone.

Pour plus d'informations sur la programmation asynchrone, consultez Use asynchronous programming dans le Guide du développeur du kit AWS SDK for Java 2.x .

Clients HTTP

Pour prendre en charge chaque client, il existe un client HTTP qui gère les communications avec les Services AWS. Vous pouvez connecter d'autres clients HTTP en choisissant celui qui présente les caractéristiques les mieux adaptées à votre application. Certains sont plus légers ; d'autres offrent davantage d'options de configuration.

Certains clients HTTP ne prennent en charge que l'utilisation synchrone, tandis que d'autres prennent uniquement en charge l'utilisation asynchrone. Pour un organigramme qui peut vous aider à sélectionner le client HTTP optimal pour votre charge de travail, consultez HTTP client recommendations dans le Guide du développeur du kit AWS SDK for Java 2.x .

La liste suivante présente certains des clients HTTP possibles :

Rubriques

- Client HTTP basé sur Apache
- Client HTTP basé sur URLConnection
- Client HTTP basé sur Netty
- AWS Client HTTP basé sur CRT

Client HTTP basé sur Apache

La classe `ApacheHttpClient` prend en charge les clients de service synchrones. Il s'agit du client HTTP par défaut pour une utilisation synchrone. Pour en savoir plus sur la configuration de la classe `ApacheHttpClient`, consultez Configuration du client HTTP basé sur Apache dans le Guide du développeur du kit AWS SDK for Java 2.x .

Client HTTP basé sur **`URLConnection`**

La classe `UrlConnectionHttpClient` est une autre option pour les clients synchrones. Elle se charge plus rapidement que le client HTTP basé sur Apache, mais possède moins de

fonctionnalités. Pour plus d'informations sur la configuration `UrlConnectionHttpClient` de la classe, voir Configurer le client HTTP URLConnection basé dans le Guide du AWS SDK for Java 2.x développeur.

Client HTTP basé sur Netty

La classe `NettyNioAsyncHttpClient` prend en charge les clients asynchrones. C'est le choix par défaut pour une utilisation asynchrone. Pour en savoir plus sur la configuration de la classe `NettyNioAsyncHttpClient`, consultez Configure the Netty-based HTTP client dans le Guide du développeur du kit AWS SDK for Java 2.x .

AWS Client HTTP basé sur CRT

Les nouvelles `AwsCrtAsyncHttpClient` classes `AwsCrtHttpClient` et les classes des bibliothèques AWS Common Runtime (CRT) sont davantage des options qui prennent en charge les clients synchrones et asynchrones. Comparé aux autres clients HTTP, AWS CRT propose :

- Temps de démarrage plus rapide du kit SDK
- Empreinte mémoire réduite
- Temps de latence réduit
- Gestion de l'état des connexions
- Équilibrage de charge DNS

Pour plus d'informations sur la configuration `AwsCrtAsyncHttpClient` des classes `AwsCrtHttpClient` et, voir Configurer les clients HTTP AWS basés sur CRT dans le Guide du AWS SDK for Java 2.x développeur.

Le client HTTP AWS CRT n'est pas le client par défaut car cela romprait la rétrocompatibilité des applications existantes. Toutefois, pour DynamoDB, nous vous recommandons d'utiliser AWS le client HTTP CRT pour les utilisations synchronisées et asynchrones.

Pour une présentation du client HTTP AWS CRT, voir Annonce de la disponibilité du client HTTP AWS CRT AWS SDK for Java 2.x sur le blog consacré aux outils de AWS développement.

Configuration d'un client HTTP

Lorsque vous configurez un client, vous pouvez proposer différentes options de configuration, notamment :

- Définition de délais d'expiration pour différents aspects des appels d'API.
- Activation de TCP Keep-Alive.
- Contrôle de la politique de nouvelles tentatives en cas d'erreur.
- Spécification des attributs d'exécution que les instances d'intercepteur d'exécution peuvent modifier. Les intercepteurs d'exécution peuvent écrire du code qui intercepte l'exécution des demandes et réponses d'API. Cela vous permet d'effectuer des tâches telles que la publication de métriques et la modification des demandes en cours de route.
- Ajout ou manipulation des en-têtes HTTP.
- Activation du suivi des métriques de performance côté client. L'utilisation de cette fonctionnalité vous permet de collecter des statistiques sur les clients du service dans votre application et d'analyser les résultats sur Amazon CloudWatch.
- Spécification d'un autre service d'exécuteur à utiliser pour planifier les tâches, telles que les nouvelles tentatives asynchrones et les tâches de délai d'expiration.

Vous contrôlez la configuration en fournissant un objet `ClientOverrideConfiguration` à la classe `Builder` du client de service. Vous le verrez dans certains exemples de code dans les sections suivantes.

La `ClientOverrideConfiguration` fournit des choix de configuration standard. Les différents clients HTTP enfichables offrent également des possibilités de configuration spécifiques à l'implémentation.

Rubriques de cette section

- Configuration du délai d'attente
- RetryMode
- DefaultsMode
- Configuration de Keep-Alive

Configuration du délai d'attente

Vous pouvez ajuster la configuration du client pour contrôler les différents délais d'expiration liés aux appels de service. DynamoDB fournit des latences plus faibles que les autres Services AWS. Par conséquent, vous souhaiterez peut-être ajuster ces propriétés pour réduire les valeurs de délai d'expiration afin de pouvoir procéder à une interruption immédiate en cas de problème réseau.

Vous pouvez personnaliser le comportement lié à la latence à l'aide de la `ClientOverrideConfiguration` du client DynamoDB ou en modifiant les options de configuration détaillées sur l'implémentation du client HTTP sous-jacent.

Vous pouvez configurer les propriétés percutantes suivantes à l'aide de la `ClientOverrideConfiguration` :

- `apiCallAttemptTimeout` : temps d'attente d'une seule tentative pour qu'une demande HTTP soit terminée avant d'abandonner et de dépasser le délai imparti.
- `apiCallTimeout` : durée dont dispose le client pour exécuter complètement un appel d'API. Cela inclut l'exécution du gestionnaire de demandes qui comprend toutes les demandes HTTP, y compris les nouvelles tentatives.

AWS SDK for Java 2.x fournit des valeurs par défaut pour certaines options de temporisation, telles que le délai de connexion et le délai d'expiration du socket. Le kit SDK ne fournit pas de valeurs par défaut pour les délais d'expiration d'appel d'API ou les délais d'expiration individuels des tentatives d'appel d'API. Si ces délais d'expiration ne sont pas définis dans la `ClientOverrideConfiguration`, le kit SDK utilise plutôt la valeur du délai d'expiration du socket pour le délai d'expiration global d'appel d'API. La délai d'expiration du socket a une valeur par défaut de 30 secondes.

RetryMode

Une autre configuration liée à la configuration du délai d'expiration que vous devez prendre en compte est l'objet de configuration `RetryMode`. Cet objet de configuration contient un ensemble de comportements de nouvelle tentative.

Le kit SDK pour Java 2.x est compatible avec les modes de nouvelle tentative suivants :

- `legacy` : mode de nouvelle tentative par défaut si vous ne le modifiez pas explicitement. Ce mode de nouvelle tentative est spécifique au kit Java SDK. Il se caractérise par un maximum de trois nouvelles tentatives, voire plus pour des services tels que DynamoDB, qui compte jusqu'à huit nouvelles tentatives.
- `standard`— Nommé « standard » parce qu'il est plus cohérent avec les autres AWS SDKs. Ce mode attend une durée aléatoire comprise entre 0 ms et 1 000 ms pour la première nouvelle tentative. Si une autre nouvelle tentative est nécessaire, ce mode choisit un autre temps aléatoire compris entre 0 ms et 1 000 ms et le multiplie par deux. Si une nouvelle tentative supplémentaire est nécessaire, il effectue le même choix aléatoire multiplié par quatre, et ainsi de suite. Chaque

attente est limitée à 20 secondes. Ce mode effectue de nouvelles tentatives sur un plus grand nombre de conditions de défaillance détectées que le mode `legacy` lui-même. Pour DynamoDB, il effectue jusqu'à trois tentatives maximum au total, sauf si vous remplacez la valeur par numRetries.

- `adaptive` : s'appuie sur le mode `standard` et limite dynamiquement le taux de demandes AWS afin de maximiser le taux de réussite. Cela peut se produire au détriment de la latence de demande. Nous ne recommandons pas le mode de nouvelle tentative adaptatif lorsqu'une latence prévisible est importante.

Vous trouverez une définition détaillée de ces modes de nouvelle tentative dans la rubrique Comportement des nouvelles tentatives du Guide de référence sur les outils AWS SDKs and.

Politiques de nouvelles tentatives

Toutes les configurations de `RetryMode` ont une `RetryPolicy`, qui est construite sur la base d'une ou plusieurs configurations de `RetryCondition`. Cette `TokenBucketRetryCondition` est particulièrement importante pour le comportement de nouvelle tentative de l'implémentation du client de kit SDK DynamoDB. Cette condition limite le nombre de nouvelles tentatives effectuées par le kit SDK à l'aide d'un algorithme de compartiment de jetons. Selon le mode de nouvelle tentative sélectionné, les exceptions de limitation peuvent ou non soustraire des jetons au `TokenBucket`.

Lorsqu'un client rencontre une erreur pouvant faire l'objet d'une nouvelle tentative, telle qu'une exception de limitation ou une erreur temporaire du serveur, le kit SDK fait automatiquement une nouvelle tentative de demande. Vous pouvez contrôler le nombre de nouvelles tentatives et la rapidité avec lesquelles elles ont lieu.

Lorsque vous configurez un client, vous pouvez fournir une `RetryPolicy` qui prend en charge les paramètres suivants :

- `numRetries` : nombre maximum de nouvelles tentatives qui doivent être appliquées avant qu'une demande ne soit considérée comme ayant échoué. La valeur par défaut est 8 quel que soit le mode de nouvelle tentative que vous utilisez.

  > **Warning**
  >
  > Assurez-vous de modifier cette valeur par défaut après mûre réflexion.

- `backoffStrategy` : la `BackoffStrategy` à appliquer aux nouvelles tentatives, `FullJitterBackoffStrategy` étant la stratégie par défaut. Cette stratégie applique un délai

exponentiel entre les nouvelles tentatives supplémentaires en fonction du nombre actuel de nouvelles tentatives, d'un délai de base et d'un temps de backoff maximal. Cela ajoute ensuite de la nervosité pour créer un peu de hasard. Le délai de base utilisé dans le délai exponentiel est de 25 ms quel que soit le mode de nouvelle tentative.

- `retryCondition` : la `RetryCondition` détermine si une demande doit faire l'objet d'une nouvelle tentative. Par défaut, elle effectue de nouvelles tentatives pour un ensemble spécifique de codes de statut HTTP et d'exceptions pour lesquels juge qu'il est possible d'effectuer de nouvelles tentatives. Dans la plupart des cas, la configuration par défaut devrait être suffisante.

Le code suivant fournit une politique de nouvelles tentatives alternative. Il indique un total de cinq nouvelles tentatives (six demandes au total). La première nouvelle tentative doit avoir lieu après un délai d'environ 100 ms, chaque nouvelle tentative doublant ce temps de façon exponentielle, jusqu'à un délai maximum d'une seconde.

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

DefaultsMode

Les propriétés de délai d'expiration que la `ClientOverrideConfiguration` et le `RetryMode` ne gèrent pas sont généralement configurées implicitement en spécifiant un `DefaultsMode`.

La AWS SDK for Java 2.x (version 2.17.102 ou ultérieure) a introduit le support pour. `DefaultsMode` Cette fonctionnalité fournit un ensemble de valeurs par défaut pour les paramètres configurables courants, tels que les paramètres de communication HTTP, le comportement des nouvelles tentatives, les paramètres de point de terminaison régionaux du service et potentiellement toute configuration liée au kit SDK. Lorsque vous utilisez cette fonctionnalité, vous pouvez obtenir de nouvelles valeurs de configuration par défaut adaptées aux scénarios d'utilisation courants.

Les modes par défaut sont normalisés pour tous les AWS SDKs. Le kit SDK pour Java 2.x est compatible avec les modes par défaut suivants :

- `legacy` : fournit des paramètres par défaut qui varient selon le kit AWS SDK et qui existaient avant la création du `DefaultsMode`.
- `standard` : fournit des paramètres non optimisés par défaut pour la plupart des scénarios.
- `in-region`— S'appuie sur le mode standard et inclut des paramètres adaptés aux applications qui appellent Services AWS depuis le même mode Région AWS.
- `cross-region` : s'appuie sur le mode standard et inclut des paramètres avec des délais d'expiration élevés pour les applications qui appellent les Services AWS dans une région différente.
- `mobile` : s'appuie sur le mode standard et inclut des paramètres avec des délais d'expiration élevés adaptés aux applications mobiles présentant des latences plus élevées.
- `auto` : s'appuie sur le mode standard et inclut des fonctionnalités expérimentales. Le kit SDK tente de découvrir l'environnement d'exécution afin de déterminer automatiquement les paramètres appropriés. La détection automatique est basée sur l'heuristique et ne fournit pas une précision de 100 %. Si l'environnement d'exécution ne peut pas être déterminé, le mode standard est utilisé. La détection automatique peut interroger les métadonnées de l'instance et les données utilisateur, ce qui peut introduire de la latence. Si la latence de démarrage est essentielle pour votre application, nous vous recommandons de choisir plutôt un `DefaultsMode` explicite.

Vous pouvez configurer le mode de valeurs par défaut des manières suivantes :

- Directement sur un client, par le biais de `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- Sur un profil de configuration, par le biais de la propriété du fichier de profil `defaults_mode`.
- Globalement, par le biais de la propriété système `aws.defaultsMode`.
- Globalement, par le biais de la variable d'environnement `AWS_DEFAULTS_MODE`.

> **Note**
>
> Pour tout mode autre que `legacy`, les valeurs par défaut fournies peuvent changer à mesure que les bonnes pratiques évoluent. Par conséquent, si vous utilisez un mode autre que `legacy`, nous vous encourageons à effectuer des tests lors de la mise à niveau du kit SDK.

La section Paramètres de configuration intelligente du guide de référence AWS SDKs and Tools fournit une liste des propriétés de configuration et de leurs valeurs par défaut dans les différents modes par défaut.

Vous choisissez la valeur du mode par défaut en fonction des caractéristiques de votre application et du mode avec Service AWS lequel l'application interagit.

Ces valeurs sont configurées en tenant compte d'une large sélection Services AWS de valeurs. Pour un déploiement de DynamoDB classique dans lequel les tables DynamoDB et l'application sont déployées dans une seule région, le mode par défaut `in-region` est le plus pertinent parmi les modes par défaut `standard`.

Example Configuration du client du kit SDK DynamoDB optimisée pour les appels à faible latence

L'exemple suivant ajuste les délais d'expiration afin de réduire les valeurs d'un appel DynamoDB à faible latence attendu.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

L'implémentation individuelle du client HTTP peut vous permettre de contrôler encore plus précisément le délai d'expiration et le comportement d'utilisation de la connexion. Par exemple, pour le client AWS basé sur CRT, vous pouvez l'activer`ConnectionHealthConfiguration`, ce qui permet au client de surveiller activement l'état des connexions utilisées. Pour plus d'informations, consultez la section Configuration avancée des clients HTTP AWS CRT dans le Guide du AWS SDK for Java 2.x développeur.

Configuration de Keep-Alive

L'activation du mode keep-alive permet de réduire les latences en réutilisant les connexions. Il existe deux types de keep-alive : HTTP Keep-Alive et TCP Keep-Alive.

- HTTP Keep-Alive tente de maintenir la connexion HTTPS entre le client et le serveur afin que les demandes ultérieures puissent réutiliser cette connexion. Cela permet d'éviter l'authentification

HTTPS lourde lors des demandes ultérieures. HTTP Keep-Alive est activé par défaut sur tous les clients.

- TCP Keep-Alive demande au système d'exploitation sous-jacent d'envoyer de petits paquets via la connexion socket afin de garantir que le socket est maintenu en vie et de détecter immédiatement tout abandon. Cela garantit qu'une demande ultérieure ne perdra pas de temps à essayer d'utiliser un socket abandonné. Par défaut, TCP Keep-Alive est désactivé sur tous les clients. Les exemples de code suivants montrent comment l'activer sur chaque client HTTP. Lorsqu'il est activé pour tous les clients HTTP non basés sur CRT, le mécanisme Keep-Alive réel dépend du système d'exploitation. Par conséquent, vous devez configurer des valeurs TCP Keep-Alive supplémentaires, telles que le délai d'expiration et le nombre de paquets, via le système d'exploitation. Vous pouvez le faire avec `sysctl` sous Linux ou macOS, ou en utilisant les valeurs de registre sous Windows.

Example pour activer TCP Keep-Alive sur un client HTTP basé sur Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Client HTTP basé sur **`URLConnection`**

Tout client synchrone qui utilise le client HTTP basé sur `URLConnection`, `HttpURLConnection`, ne dispose pas d'un mécanisme permettant d'activer le mode keep-alive.

Example pour activer TCP Keep-Alive sur un client HTTP basé sur Netty

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example pour activer TCP Keep-Alive sur un AWS client HTTP CRT

Avec le client HTTP AWS CRT, vous pouvez activer le protocole TCP keep-alive et contrôler la durée.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
```

```
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Lorsque vous utilisez le client DynamoDB asynchrone, vous pouvez activer TCP Keep-Alive comme indiqué dans le code suivant.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Gestion des erreurs

En ce qui concerne la gestion des exceptions, il AWS SDK for Java 2.x utilise des exceptions d'exécution (non vérifiées).

L'exception de base, qui couvre toutes les exceptions du kit SDK, est `SdkServiceException`, qui s'étend à partir de l'`RuntimeException` Java non coché. Si vous détectez cela, vous détecterez toutes les exceptions générées par le kit SDK.

`SdkServiceException` possède une sous-classe appelée `AwsServiceException`. Cette sous-classe indique tout problème de communication avec le Service AWS. Il possède une sous-classe appelée `DynamoDbException`, qui indique un problème de communication avec DynamoDB. Si vous détectez ce problème, vous détecterez toutes les exceptions liées à DynamoDB, mais aucune autre exception du kit SDK.

Vous trouverez des types d'exception plus spécifiques sous `DynamoDbException`.
Certains de ces types d'exception s'appliquent aux opérations du plan de contrôle, telles que `TableAlreadyExistsException`. D'autres s'appliquent aux opérations du plan de données. Voici un exemple d'exception courante au plan de données :

- `ConditionalCheckFailedException` : vous avez spécifié une condition dans la demande, qui a été analysée comme false. Par exemple, il se peut que vous avez essayé d'effectuer une mise à jour conditionnelle sur un élément, mais que la valeur réelle de l'attribut ne corresponde pas à la valeur attendue de la condition. Une demande qui échoue de cette manière ne fait pas l'objet d'une nouvelle tentative.

Dans d'autres situations, aucune exception spécifique n'est définie. Par exemple, lorsque vos demandes font l'objet d'une limitation, les `ProvisionedThroughputExceededException` spécifiques peuvent être générées, tandis que dans d'autres cas, les `DynamoDbException` plus génériques sont générées. Dans les deux cas, vous pouvez déterminer si la limitation est à l'origine de l'exception en vérifiant si `isThrottlingException()` renvoie `true`.

En fonction des besoins de votre application, vous pouvez détecter toute `AwsServiceException` ou instance `DynamoDbException`. Cependant, vous avez souvent besoin d'un comportement différent selon les situations. La logique utilisée pour faire face à un échec de vérification d'état est différente de celle utilisée pour gérer la limitation. Définissez les chemins exceptionnels que vous souhaitez traiter et assurez-vous de tester les chemins alternatifs. Cela vous permet de vous assurer que vous pouvez faire face à tous les scénarios pertinents.

Pour obtenir la liste des erreurs courantes que vous pouvez rencontrer, consultez Gestion des erreurs avec DynamoDB. Consultez également Erreurs courantes dans la Référence de l'API Amazon DynamoDB. La référence d'API fournit également les erreurs exactes possibles pour chaque opération d'API, par exemple pour l'opération `Query`. Pour en savoir plus sur la gestion des exceptions, consultez Exception handling for the AWS SDK for Java 2.x dans le Guide du développeur du kit AWS SDK for Java 2.x .

AWS ID de demande

Chaque demande inclut un ID de demande, qu'il peut être utile d'extraire si vous utilisez AWS Support pour diagnostiquer un problème. Chaque exception dérivée de `SdkServiceException` dispose d'une méthode `requestId()` permettant d'extraire l'ID de demande.

Logging

L'utilisation de la journalisation fournie par le kit SDK peut être utile à la fois pour détecter les messages importants provenant des bibliothèques clientes et à des fins de débogage plus approfondies. Les enregistreurs sont hiérarchiques et le kit SDK utilise `software.amazon.awssdk` comme enregistreur racine. Vous pouvez configurer le niveau avec `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL` ou `OFF`. Le niveau configuré s'applique à cet enregistreur et à la hiérarchie des enregistreurs.

Pour sa journalisation, il AWS SDK for Java 2.x utilise la façade de journalisation simple pour Java (SLF4J). Cela agit comme une couche d'abstraction autour des autres enregistreurs, et vous pouvez l'utiliser pour connecter l'enregistreur que vous préférez. Pour obtenir des instructions sur le branchement des enregistreurs, consultez le manuel d'utilisation de SLF4 J.

Chaque enregistreur a un comportement particulier. Par défaut, l'enregistreur Log4j 2.x crée un `ConsoleAppender`, qui ajoute les événements du journal à `System.out` et les valeurs par défaut au niveau du journal `ERROR`.

L' SimpleLogger enregistreur inclus dans SLF4 J affiche par défaut `System.err` et par défaut le niveau du `INFO` journal.

Nous vous recommandons de définir le niveau sur `WARN` pour `software.amazon.awssdk` pour tous les déploiements de production afin de récupérer les messages importants provenant des bibliothèques clientes du kit SDK tout en limitant la quantité de sortie.

Si SLF4 je ne trouve pas d'enregistreur compatible sur le chemin de classe (aucune liaison SLF4 J), alors l'implémentation par défaut est sans opération. Cette implémentation entraîne des messages de journalisation `System.err` expliquant que SLF4 je n'ai pas trouvé d'implémentation d'enregistreur sur le chemin de classe. Pour éviter cette situation, vous devez ajouter une implémentation d'enregistreur. Pour ce faire, vous pouvez ajouter une dépendance dans votre `pom.xml` Apache Maven sur des artefacts, tels que `org.slf4j.slf4j-simple` ou `org.apache.logging.log4j.log4j-slf4j2-imp`.

Pour plus d'informations sur la configuration de la journalisation dans le kit SDK, notamment sur l'ajout de dépendances de journalisation à la configuration de votre application, consultez Logging with the SDK for Java 2.x dans le Guide du développeur du kit AWS SDK pour Java .

La configuration suivante du fichier `Log4j2.xml` montre comment ajuster le comportement de journalisation si vous utilisez l'enregistreur Apache Log4j 2. Cette configuration définit le niveau de l'enregistreur racine sur `WARN`. Tous les enregistreurs de la hiérarchie héritent de ce niveau de journalisation, y compris l'enregistreur `software.amazon.awssdk`.

Par défaut, la sortie passe à `System.out`. Dans l'exemple suivant, nous remplaçons toujours l'appender Log4j de sortie par défaut pour appliquer un `PatternLayout` Log4j adapté.

Exemple de fichier de configuration **`Log4j2.xml`**

La configuration suivante enregistre les messages sur la console aux niveaux `ERROR` et `WARN` pour toutes les hiérarchies d'enregistreurs.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
```

```
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

AWS journalisation de l'ID de demande

En cas de problème, vous pouvez trouver une demande IDs dans certaines exceptions. Toutefois, si vous souhaitez que la demande IDs concerne des demandes qui ne génèrent pas d'exceptions, vous pouvez utiliser la journalisation.

L'`software.amazon.awssdk.request`enregistreur produit la demande IDs au `DEBUG` niveau. L'exemple suivant étend l'configuration example précédent pour maintenir le niveau d'enregistreur racine sur `ERROR`, sur `software.amazon.awssdk` au niveau `WARN`, et sur `software.amazon.awssdk.request` au niveau `DEBUG`. La définition de ces niveaux permet de récupérer la demande IDs et d'autres détails liés à la demande, tels que le point de terminaison et le code d'état.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

Voici un exemple de la sortie du journal:

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
```

```
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
 2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
 successful response: 200, Request ID:
 QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
 available
```

Pagination

Certaines demandes, telles que `Query` et `Scan`, limitent la taille des données renvoyées lors d'une seule demande et nécessitent que vous fassiez des demandes répétées pour extraire les pages suivantes.

Vous pouvez contrôler le nombre maximum d'éléments à lire pour chaque page à l'aide du paramètre `Limit`. Par exemple, vous pouvez utiliser le paramètre `Limit` pour récupérer uniquement les 10 derniers éléments. Cette limite indique le nombre d'éléments à lire dans la table avant qu'un filtrage ne soit appliqué. Si vous voulez exactement 10 éléments après le filtrage, il n'y a aucun moyen de le spécifier. Vous ne pouvez contrôler que le nombre préfiltré et vérifier côté client lorsque vous avez effectivement extrait 10 éléments. Quelle que soit la limite, les réponse ont toujours une taille maximale de 1 Mo.

Une `LastEvaluatedKey` peut être incluse dans la réponse de l'API. Cela indique que la réponse s'est terminée parce qu'elle a atteint une limite de nombre ou de taille. La clé est la dernière clé évaluée pour la réponse. En interagissant directement avec l'API, vous pouvez le extraire cette `LastEvaluatedKey` et la transmettre à un appel de suivi en tant que `ExclusiveStartKey` afin de lire le fragment suivant à partir de ce point de départ. Si aucune `LastEvaluatedKey` n'est renvoyée, cela signifie qu'il n'y a plus d'éléments correspondant à l'appel d'API `Query` ou `Scan`.

L'exemple suivant utilise l'interface de bas niveau pour limiter les éléments à 100 en fonction du paramètre `keyConditionExpression`.

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());
```

```
    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

Ils AWS SDK for Java 2.x peuvent simplifier cette interaction avec DynamoDB en fournissant des méthodes de pagination automatique qui effectuent plusieurs appels de service pour obtenir automatiquement les pages de résultats suivantes pour vous. Cela simplifie le code, mais vous prive d'un certain contrôle de l'utilisation des ressources que vous pourriez conserver en lisant les pages manuellement.

En utilisant les méthodes `Iterable` disponibles dans le client DynamoDB, telles que `QueryPaginator` et `ScanPaginator`, le kit SDK prend en charge la pagination. Le type de retour de ces méthodes est un itérable personnalisé que vous pouvez utiliser pour itérer sur toutes les pages. Le kit SDK gère les appels de service en interne pour vous. À l'aide de l'API Java Stream, vous pouvez gérer le résultat de `QueryPaginator` comme illustré dans l'exemple suivant.

```
QueryPublisher queryPublisher =
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

Annotations de classes de données

Le kit SDK Java fournit plusieurs annotations que vous pouvez ajouter aux attributs de la classe de données. Ces annotations influencent la manière dont le kit SDK interagit avec les attributs. En

ajoutant une annotation, vous pouvez faire en sorte qu'un attribut se comporte comme un compteur atomique implicite, conserver une valeur d'horodatage générée automatiquement ou suivre le numéro de version d'un article. Pour plus d'informations, consultez Annotations de classes de données.

Gestion des erreurs avec DynamoDB

Cette section décrit les erreurs d'exécution et la façon de les traiter. Elle décrit également les messages d'erreur et les codes spécifiques d'Amazon DynamoDB. Pour obtenir la liste des erreurs courantes qui s'appliquent à tous les AWS services, consultez la section Gestion des accès

Rubriques

- Composants erreur
- Erreurs transactionnelles
- Codes et messages d'erreur
- Gestion des erreurs dans votre application
- Nouvelles tentatives après erreur et backoff exponentiel
- Opérations par lot et gestion des erreurs

Composants erreur

Quand votre programme envoie une demande, DynamoDB tente de la traiter. Si la demande aboutit, DynamoDB renvoie un code d'état HTTP de succès (`200 OK`), ainsi que les résultats de l'opération demandée.

Si la demande échoue, DynamoDB renvoie une erreur. Chaque erreur possède trois composants :

- Un code d'état HTTP (`400`, par exemple).
- Un nom d'exception (`ResourceNotFoundException`, par exemple).
- Un message d'erreur (`Requested resource not found: Table: tablename not found`, par exemple).

AWS SDKs Prenez soin de propager les erreurs dans votre application afin que vous puissiez prendre les mesures appropriées. Par exemple, dans un programme Java, vous pouvez écrire une logique `try-catch` de façon à gérer un `ResourceNotFoundException`.

Si vous n'utilisez pas de AWS SDK, vous devez analyser le contenu de la réponse de bas niveau de DynamoDB. Voici un exemple de réponse.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

Erreurs transactionnelles

Pour plus d'informations sur les erreurs transactionnelles, veuillez consulter Gestion des conflits de transactions dans DynamoDB

Codes et messages d'erreur

Voici la liste des exceptions renvoyées par DynamoDB, regroupées par code d'état HTTP. Si OK to retry? a pour réponse Yes, vous pouvez soumettre à nouveau la même demande. Si OK to retry? a pour réponse No, vous devez résoudre le problème côté client avant de soumettre une nouvelle demande.

Code d'état HTTP 400

Un code d'état HTTP 400 indique un problème avec votre demande, tel qu'un échec d'authentification, l'absence de paramètres requis ou le dépassement du débit provisionné d'une table. Vous devez corriger le problème dans votre application avant de soumettre à nouveau la demande.

AccessDeniedException

Message : Accès refusé.

Le client n'a pas signé correctement la demande. Si vous utilisez un kit AWS SDK, les demandes sont signées automatiquement ; sinon, accédez au Processus de signature Signature Version 4 dans le Références générales AWS.

OK pour réessayer ? Non

ConditionalCheckFailedException

Message : Échec de la demande conditionnelle.

Vous avez spécifié une condition qui a été analysée comme false. Par exemple, il se peut que vous avez essayé d'effectuer une mise à jour conditionnelle sur un élément, mais que la valeur réelle de l'attribut ne corresponde pas à la valeur attendue de la condition.

OK pour réessayer ? Non

IncompleteSignatureException

Message : La signature de la demande n'est pas conforme aux normes AWS.

La signature de la demande ne comprenait pas tous les composants requis. Si vous utilisez un AWS SDK, les demandes sont signées automatiquement pour vous ; sinon, passez au processus de signature Signature version 4 dans le Références générales AWS.

OK pour réessayer ? Non

ItemCollectionSizeLimitExceededException

Message : Dépassement de taille de la collection.

Pour une table avec un index secondaire local, un groupe d'éléments avec la même valeur de clé de partition a dépassé la limite de taille maximum de 10 Go. Pour plus d'informations sur les collections d'éléments, consultez Collections d'articles dans les index secondaires locaux.

OK pour réessayer ? Oui

LimitExceededException

Message : Beaucoup trop d'opérations pour un abonné donné.

Il y a beaucoup trop d'opérations de contrôle simultanées. Le nombre cumulé de tables et d'index à l'état `CREATING`, `DELETING` ou `UPDATING` ne peut pas dépasser 500.

OK pour réessayer ? Oui

MissingAuthenticationTokenException

Message : La demande doit comporter un ID de clé d'accès AWS valide (inscrit).

La demande n'incluait pas l'en-tête d'autorisation obligatoire ou n'était pas correctement formée. Consultez API de bas niveau de DynamoDB.

OK pour réessayer ? Non

ProvisionedThroughputExceededException

Message : Vous avez dépassé le débit provisionné autorisé maximal pour une table ou pour un ou plusieurs index secondaires globaux. Pour consulter les indicateurs de performance relatifs au débit provisionné par rapport au débit consommé, ouvrez la console Amazon. CloudWatch

L'erreur inclut une liste de champs `ThrottlingReason` fournissant un contexte spécifique expliquant pourquoi la limitation s'est produite, en fonction du format `ResourceType+OperationType+LimitType` (par exemple, `TableReadProvisionedThroughputExceeded`) et de l'ARN de la ressource affectée. Cela vous permet d'identifier la ressource limitée (table ou index), le type d'opération qui a déclenché la limitation (lecture ou écriture) et la limite spécifique qui a été dépassée (dans ce cas, la capacité allouée). Pour plus d'informations sur le diagnostic et la résolution des problèmes de limitation, consultez Diagnostic des problèmes de limitation.

Exemple : votre taux de demandes est trop élevé.AWS SDKs Pour DynamoDB, réessayez automatiquement les demandes qui reçoivent cette exception. Votre demande finit par aboutir, à moins que la file d'attente des nouvelles tentatives soit trop grande pour être terminée. Réduisez la fréquence des demandes avec Nouvelles tentatives après erreur et backoff exponentiel.

OK pour réessayer ? Oui

ReplicatedWriteConflictException

Message : Un ou plusieurs éléments de cette demande sont modifiés par une demande dans une autre région.

Exemple : Une opération d'écriture a été demandée pour un élément d'une table globale multirégionale fortement cohérente (MRSC) qui est actuellement modifiée par une demande dans une autre région.

OK pour réessayer ? Oui

RequestLimitExceeded

Message : Throughput exceeds the current throughput limit for your account (le débit dépasse la limite de débit actuelle de votre compte). Pour demander une augmentation de limite, contactez le AWS Support à l'adresse https://aws.amazon.com/support.

L'erreur inclut une liste de champs `ThrottlingReason` fournissant un contexte spécifique expliquant pourquoi la limitation s'est produite, en fonction du format `ResourceType+OperationType+LimitType` (par exemple, `TableWriteAccountLimitExceeded` ou `IndexReadAccountLimitExceeded`) et de l'ARN de la ressource affectée. Cela vous permet d'identifier quelle ressource est limitée (table ou index), quel type d'opération a déclenché la limitation (lecture ou écriture) et si vous avez dépassé les quotas de service au niveau du compte. Pour plus d'informations sur le diagnostic et la résolution des problèmes de limitation, consultez Diagnostic des problèmes de limitation.

Exemple : Le taux de demandes à la demande dépasse le débit autorisé du compte et la table ne peut pas être davantage mise à l'échelle.

OK pour réessayer ? Oui

ResourceInUseException

Message : La ressource que vous essayez de modifier est en cours d'utilisation.

Exemple : vous avez essayé de recréer une table existante ou de supprimer une table dont l'état est `CREATING`.

OK pour réessayer ? Non

ResourceNotFoundException

Message : La ressource demandée est introuvable.

Exemple : la table demandée n'existe pas ou se trouve trop tôt dans l'état `CREATING`.

OK pour réessayer ? Non

ThrottlingException

Message : Le taux de demandes dépasse le débit autorisé.

Cette exception est renvoyée sous forme de AmazonServiceException réponse avec un code d'état THROTTLING_EXCEPTION. Cette exception peut être renvoyée si vous effectuez des opérations d'API de plan de contrôle trop rapidement.

Pour les tables utilisant le mode à la demande, cette exception peut être renvoyée pour toute opération d'API de plan de données si votre taux de demandes est trop élevé. Pour en savoir plus sur la mise à l'échelle à la demande, consultez Débit initial et propriétés de mise à l'échelle.

L'erreur inclut une liste de champs `ThrottlingReason` fournissant un contexte spécifique expliquant pourquoi la limitation s'est produite, en fonction du format `ResourceType +OperationType+LimitType` (par exemple, `TableReadKeyRangeThroughputExceeded` ou `IndexWriteMaxOnDemandThroughputExceeded`) et de l'ARN de la ressource affectée. Ces informations vous permettent d'identifier la ressource limitée (table ou index), le type d'opération qui a déclenché la limitation (lecture ou écriture) et la limite spécifique qui a été dépassée (limites de partition ou débit maximum à la demande). Pour plus d'informations sur le diagnostic et la résolution des problèmes de limitation, consultez Diagnostic des problèmes de limitation.

OK pour réessayer ? Oui

UnrecognizedClientException

Message : L'ID de clé d'accès ou le jeton de sécurité n'est pas valide.

La signature de la demande est incorrecte. La cause la plus probable est un identifiant de cléAWS d'accès ou une clé secrète non valide.

OK pour réessayer ? Oui

ValidationException

Message : variable selon les erreurs spécifiques rencontrées

Cette erreur peut se produire pour différentes raisons, telles qu'un paramètre obligatoire absent, une valeur en dehors des limites ou une incompatibilité des données. Le message d'erreur contient des détails sur la partie spécifique de la demande qui a provoqué l'erreur.

OK pour réessayer ? Non

Code d'état HTTP 5xx

Un code d'état HTTP `5xx` indique un problème qui doit être résolu par AWS. Il peut s'agir d'une erreur temporaire, auquel cas vous pouvez retenter votre demande jusqu'à ce qu'elle aboutisse. Sinon, accédez à l'AWS Service Health Dashboard pour voir si le service rencontre des problèmes opérationnels.

Pour plus d'informations, consultez l'article Comment résoudre les erreurs HTTP 5xx dans Amazon DynamoDB ?

InternalServerError (HTTP 500)

DynamoDB n'a pas pu traiter votre demande.

OK pour réessayer ? Oui

> **Note**
>
> Vous pouvez rencontrer des erreurs de serveur internes lorsque vous travaillez avec des éléments. Celles-ci sont attendues pendant la durée de vie d'une table. Toute demande en échec peut être relancée immédiatement.
> Lorsque vous recevez un code d'état 500 lors d'une opération d'écriture, l'opération peut avoir réussi ou échoué. Si l'opération d'écriture est une demande `TransactWriteItem`, il convient de réessayer l'opération. Si l'opération d'écriture est une demande d'écriture à élément unique`PutItem`, telle que`UpdateItem`, ou`DeleteItem`, alors votre application doit lire l'état de l'élément avant de recommencer l'opération, and/or Exemple de commande CLI d'expression de condition DynamoDB afin de s'assurer que l'élément reste dans un état correct après une nouvelle tentative, que l'opération précédente ait réussi ou échoué. Si l'idempotence est une exigence pour l'opération d'écriture, veuillez utiliser `TransactWriteItem`, qui prend en charge les demandes idempotentes en spécifiant automatiquement un `ClientRequestToken` pour désambiguïser plusieurs tentatives d'exécution de la même action.

ServiceUnavailable (HTTP 503)

DynamoDB est actuellement indisponible. (Il doit s'agir d'un état temporaire.)

OK pour réessayer ? Oui

Gestion des erreurs dans votre application

Pour que votre application fonctionne correctement, vous devez ajouter une logique destinée à récupérer les erreurs et à y répondre. Les approches classiques incluent l'utilisation de blocs `try-catch` ou d'instructions `if-then`.

Ils AWS SDKs effectuent leurs propres tentatives et vérifient les erreurs. Si vous rencontrez une erreur lors de l'utilisation de l'un d'entre eux AWS SDKs, le code d'erreur et sa description peuvent vous aider à résoudre le problème.

Vous devez également voir un `Request ID` dans la réponse. Cela `Request ID` peut être utile si vous devez contacter le AWS Support pour diagnostiquer un problème.

Nouvelles tentatives après erreur et backoff exponentiel

Un grand nombre de composants d'un réseau, tels que serveurs DNS, commutateurs, équilibreurs de charge ou autres, peuvent générer des erreurs à n'importe quel moment de la vie d'une requête donnée. La technique habituelle pour traiter ces réponses erronées dans un environnement réseau consiste à implémenter les nouvelles tentatives dans l'application cliente. Cette technique augmente la fiabilité de l'application.

Chaque AWS SDK implémente automatiquement une logique de nouvelle tentative. Vous pouvez modifier les paramètres de nouvelle tentative en fonction de vos besoins. Par exemple, imaginons une application Java nécessitant une politique d'interruption immédiate, sans autorisation de nouvelle tentative en cas d'erreur. Avec le AWS SDK pour Java, vous pouvez utiliser la `ClientConfiguration` classe et fournir une `maxErrorRetry` valeur de `0` pour désactiver les nouvelles tentatives. Pour plus d'informations, consultez la documentation du AWS SDK correspondant à votre langage de programmation.

Si vous n'utilisez pas de AWS SDK, vous devez réessayer les demandes d'origine qui ont généré des erreurs de serveur (5xx). Cependant, les erreurs client (4xx, autres qu'une exception `ThrottlingException` ou `ProvisionedThroughputExceededException`) indiquent que vous avez besoin de réviser la demande elle-même pour résoudre le problème avant de recommencer. Pour obtenir des recommandations détaillées concernant des scénarios de limitation spécifiques, reportez-vous à la section de résolution des problèmes de limitation de DynamoDB.

Outre de simples tentatives, chaque AWS SDK implémente un algorithme de ralentissement exponentiel pour un meilleur contrôle du flux. Le concept sous-jacent vise à utiliser des temps

d'attente progressivement plus longs entre les tentatives en cas de réponses d'erreur consécutives. Par exemple, jusqu'à 50 millisecondes avant la première nouvelle tentative, jusqu'à 100 millisecondes avant la deuxième, jusqu'à 200 millisecondes avant la troisième, et ainsi de suite. Cependant, après une minute, si la demande a échoué, le problème peut être que la taille de la demande entraîne un dépassement du débit provisionné, et ne pas être lié au taux de demandes. Définir le nombre maximal de nouvelles tentatives pour qu'elles s'arrêtent au bout d'une minute environ. Si la demande n'aboutit pas, examinez vos options de débit provisionné.

> **Note**
>
> Ils AWS SDKs implémentent une logique de nouvelle tentative automatique et un ralentissement exponentiel.

La plupart des algorithmes de backoff exponentiel utilisent l'instabilité (retard aléatoire) pour éviter les conflits successifs. Comme vous ne cherchez pas à éviter de tels conflits dans ces cas-là, vous n'avez pas besoin d'utiliser ce nombre aléatoire. Cependant, si vous utilisez les clients simultanés, l'instabilité peut aider à ce que vos demandes réussissent plus rapidement. Pour plus d'informations, consultez le billet de blog sur Exponential backoff and jitter.

Opérations par lot et gestion des erreurs

L'API de bas niveau DynamoDB prend en charge les opérations par lot pour les lectures et les écritures. `BatchGetItem` lit les éléments d'une ou de plusieurs tables, et `BatchWriteItem` insère ou supprime des éléments dans une ou plusieurs tables. Ces opérations par lot sont implémentées sous la forme d'encapsuleurs autour des opérations DynamoDB autres que par lot. En d'autres termes, `BatchGetItem` appelle `GetItem` une fois pour chaque élément du lot. De même, `BatchWriteItem` invoque `DeleteItem` ou `PutItem`, le cas échéant, pour chaque élément du lot.

Une opération par lot peut supporter la défaillance de requêtes individuelles dans le lot. Par exemple, imaginons une demande `BatchGetItem` de lecture de cinq éléments. Même si certaines demandes `GetItem` sous-jacentes échouent, il ne s'ensuit pas que l'ensemble de l'opération `BatchGetItem` échoue. Cependant, si les cinq opérations de lecture échouent, alors `BatchGetItem` échoue dans son intégralité.

Les opérations par lot renvoient des informations sur les demandes individuelles qui échouent pour vous permettre de diagnostiquer le problème et de réessayer l'opération. Pour `BatchGetItem`, les tables et les clés primaires en question sont retournées dans la valeur `UnprocessedKeys`

de la réponse. Pour `BatchWriteItem`, les informations similaires sont retournées dans `UnprocessedItems`.

La cause la plus probable d'un échec en lecture ou en écriture est la limitation. Pour `BatchGetItem`, une ou plusieurs tables de la demande par lot n'ont pas une capacité en lecture suffisamment provisionnée pour prendre en charge l'opération. Pour `BatchWriteItem`, une ou plusieurs des tables n'ont pas une capacité en écriture suffisamment provisionnée.

Si DynamoDB renvoie des éléments non traités, vous devez relancer l'opération par lot sur ces éléments. Cependant, nous vous recommandons vivement d'utiliser un algorithme d'interruption exponentielle. Si vous recommencez immédiatement l'opération par lot, les demandes en lecture ou en écriture sous-jacentes peuvent continuer à échouer en raison de la limitation sur les tables individuelles. Si vous retardez l'opération par lot à l'aide d'une interruption exponentielle, il est très vraisemblable que les demandes du lot réussiront.

Utilisation de DynamoDB avec un SDK AWS

AWS des kits de développement logiciel (SDKs) sont disponibles pour de nombreux langages de programmation courants. Chaque kit SDK fournit une API, des exemples de code et de la documentation qui facilitent la création d'applications par les développeurs dans leur langage préféré.

Documentation SDK	Exemples de code
AWS SDK pour C++	AWS SDK pour C++ exemples de code
AWS CLI	AWS CLI exemples de code
AWS SDK pour Go	AWS SDK pour Go exemples de code
AWS SDK pour Java	AWS SDK pour Java exemples de code
AWS SDK pour JavaScript	AWS SDK pour JavaScript exemples de code
AWS SDK pour Kotlin	AWS SDK pour Kotlin exemples de code
AWS SDK pour .NET	AWS SDK pour .NET exemples de code
AWS SDK pour PHP	AWS SDK pour PHP exemples de code
Outils AWS pour PowerShell	Outils AWS pour PowerShell exemples de code

Documentation SDK	Exemples de code
AWS SDK pour Python (Boto3)	AWS SDK pour Python (Boto3) exemples de code
AWS SDK pour Ruby	AWS SDK pour Ruby exemples de code
AWS SDK pour Rust	AWS SDK pour Rust exemples de code
AWS SDK pour SAP ABAP	AWS SDK pour SAP ABAP exemples de code
AWS SDK pour Swift	AWS SDK pour Swift exemples de code

Pour accéder à des exemples spécifiques à DynamoDB, consultez Exemples de code pour DynamoDB utilisant AWS SDKs.

> **Exemple de disponibilité**
>
> Vous n'avez pas trouvé ce dont vous avez besoin ? Demandez un exemple de code en utilisant le lien Provide feedback (Fournir un commentaire) en bas de cette page.

Utilisation de tables, d'éléments, de requêtes, d'analyses et d'index

Cette section fournit des détails sur l'utilisation de tables, d'éléments, de requêtes et bien plus dans Amazon DynamoDB.

Rubriques

- Utilisation de tables et de données dans DynamoDB
- Tables globales : réplication multi-active, multirégion
- Utilisation d'éléments et d'attributs dans DynamoDB
- Amélioration de l'accès aux données avec les index secondaires dans DynamoDB
- Gestion des flux de travail complexes avec des transactions DynamoDB
- Récupération de données de modification avec Amazon DynamoDB

Utilisation de tables et de données dans DynamoDB

Cette section explique comment créer, mettre à jour et supprimer des tables dans Amazon DynamoDB à l'aide de l'AWS Command Line Interface (AWS CLI) et de kits SDK AWS.

> **Note**
>
> Vous pouvez également effectuer ces tâches à l'aide de AWS Management Console. Pour en savoir plus, consultez Utilisation de la console.

Cette section fournit également des informations supplémentaires sur la capacité de débit, l'utilisation de l'autoscaling de DynamoDB ou le paramétrage manuel du débit alloué.

Rubriques

- Opérations de base sur les tables DynamoDB
- Points à prendre en considération lors du choix d'une classe de tables dans DynamoDB
- Ajout de balises et d'étiquettes aux ressources dans DynamoDB

Opérations de base sur les tables DynamoDB

Comme d'autres systèmes de base de données, Amazon DynamoDB stocke les données dans des tables. Vous pouvez gérer vos tables à l'aide de quelques opérations de base.

Rubriques

- Création d'une table
- Description d'une table
- Mise à jour d'une table
- Suppression d'une table
- Utilisation de la protection contre la suppression
- Liste des noms de table
- Description des quotas de débit alloué

Création d'une table

L'opération `CreateTable` vous permet de créer une table dans Amazon DynamoDB. Pour créer la table, vous devez fournir les informations suivantes :

- Nom de la table. Le nom doit être conforme aux règles de dénomination DynamoDB et unique pour le compte et la région AWS actuels. Par exemple, vous pouvez créer une table `People` dans la région USA Est (Virginie du Nord), et une autre table `People` en Europe (Irlande). Ces deux tables seraient toutefois entièrement différentes l'une de l'autre. Pour en savoir plus, consultez Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.
- Clé primaire. La clé primaire peut se composer d'un attribut (clé de partition) ou de deux attributs (clé de partition et clé de tri). Vous devez indiquer les noms d'attribut, les types de données et le rôle de chaque attribut : `HASH` (pour une clé de partition) et `RANGE` (pour une clé de tri). Pour en savoir plus, consultez Clé primaire.
- Paramètres de débit (pour les tables allouées) Si vous utilisez le mode alloué, vous devez spécifier les paramètres de débit initial de lecture et d'écriture pour la table. Vous pouvez modifier ces paramètres ultérieurement ou activer l'autoscaling de DynamoDB afin que les paramètres soient gérés pour vous. Pour en savoir plus, consultez Mode de capacité provisionnée DynamoDB et Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

Exemple 1 : créer une table à la demande

Pour créer la même table `Music` en mode à la demande.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

L'opération `CreateTable` renvoie des métadonnées pour la table, comme illustré ci-dessous.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
```

```
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397468.348
    }
}
```

> **Important**
>
> Lorsque vous appelez `DescribeTable` sur une table à la demande, les unités de capacité de lecture et d'écriture sont remises à 0.

Exemple 2 : créer une table allouée

L'exemple AWS CLI suivant montre comment créer une table (`Music`). La clé primaire se compose de `Artist` (clé de partition) et de `SongTitle` (clé de tri), chacune d'elles ayant le type de données `String`. Le débit maximum pour cette table est de 10 unités de capacité de lecture et 5 unités de capacité d'écriture.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

L'opération `CreateTable` renvoie des métadonnées pour la table, comme illustré ci-dessous.

```
{
    "TableDescription": {
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

L'élément `TableStatus` indique l'état réel de la table (`CREATING`). La création de la table peut prendre un certain temps, selon les valeurs que vous spécifiez pour `ReadCapacityUnits` et `WriteCapacityUnits`. Plus ces valeurs sont élevées, plus DynamoDB doit allouer des ressources à la table.

Exemple 3 : créer une table à l'aide de la classe de tables DynamoDB Standard-Infrequent Access

Pour créer la même table `Music` à l'aide de la classe de tables DynamoDB Standard-Infrequent Access.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

L'opération `CreateTable` renvoie des métadonnées pour la table, comme illustré ci-dessous.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
```

```
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

Description d'une table

Affichez les détails d'une table à l'aide de l'opération `DescribeTable`. Vous devez fournir le nom de la table. La sortie de `DescribeTable` est au même format que celle de `CreateTable`. Elle inclut l'horodatage de la création de la table, son schéma de clé, ses paramètres de débit alloué, sa taille estimée et tout index secondaire présent.

> **Important**
>
> Lorsque vous appelez `DescribeTable` sur une table à la demande, les unités de capacité de lecture et d'écriture sont remises à 0.

Example

```
aws dynamodb describe-table --table-name Music
```

La table est prête à l'emploi lorsque `TableStatus` passe de `CREATING` à `ACTIVE`.

> **Note**
>
> Si vous exécutez une demande `DescribeTable` immédiatement après une demande `CreateTable`, DynamoDB peut renvoyer une erreur (`ResourceNotFoundException`). En effet, `DescribeTable` utilise une demande à cohérence à terme et les métadonnées pour

votre table peuvent ne pas être disponibles à ce moment-là. Attendez quelques secondes, puis réessayez d'envoyer la demande `DescribeTable`.
À des fins de facturation, vos coûts de stockage DynamoDB incluent une surcharge par élément de 100 octets. (Pour en savoir plus, consultez Tarification DynamoDB.) Ces 100 octets supplémentaires par élément ne sont pas utilisés dans les calculs d'unité de capacité ou par l'opération `DescribeTable`.

Mise à jour d'une table

L'opération `UpdateTable` vous permet d'exécuter l'une des actions suivantes :

- Modifier les paramètres de débit alloué d'une table (pour les tables en mode alloué)
- Changer le mode de capacité de lecture ou d'écriture de la table
- Manipuler des index secondaires globaux dans la table (voir Utilisation d'index secondaires globaux dans DynamoDB)
- Activer ou désactiver DynamoDB Streams sur la table (voir Modifier la récupération de données pour DynamoDB Streams)

Example

L'exemple d'AWS CLI suivant illustre comment modifier les paramètres de débit alloué d'une table.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> Lorsque vous émettez une demande `UpdateTable`, le statut de la table passe de `AVAILABLE` à `UPDATING`. La table reste entièrement utilisable pendant sa `UPDATING`. Lorsque ce processus est terminé, le statut de la table passe de `UPDATING` à `AVAILABLE`.

Example

L'exemple d'AWS CLI suivant montre comment modifier le mode de capacité de lecture ou d'écriture pour activer le mode à la demande.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Suppression d'une table

Vous pouvez supprimer une table non utilisée avec l'opération `DeleteTable`. Une fois supprimée, une table est irrécupérable. Pour supprimer une table à l'aide d'AWS Management Console, consultez the section called "Étape 6 : nettoyage (facultatif)".

Example

L'exemple d'AWS CLI suivant illustre la suppression d'une table.

```
aws dynamodb delete-table --table-name Music
```

Lorsque vous émettez une demande `DeleteTable`, le statut de la table passe de `ACTIVE` à `DELETING`. La suppression de la table peut prendre un certain temps, selon les ressources utilisées (par exemple les données stockées dans la table ou les flux ou index sur la table).

Une fois l'opération `DeleteTable` terminée, la table n'existe plus dans DynamoDB.

Utilisation de la protection contre la suppression

Vous pouvez protéger une table contre toute suppression accidentelle grâce à la propriété de protection contre la suppression. L'activation de cette propriété pour les tables permet de s'assurer que celles-ci ne seront pas supprimées accidentellement lors des opérations classiques de gestion des tables par vos administrateurs. Cela vous aidera à éviter toute interruption de vos activités métier normales.

Le propriétaire de la table ou un administrateur autorisé contrôle la propriété de protection contre la suppression pour chaque table. La propriété de protection contre la suppression pour chaque table est désactivée par défaut. Cela inclut les répliques globales et les tables restaurées à partir de sauvegardes. Lorsque la protection contre la suppression est désactivée pour une table, celle-ci peut être supprimée par tous les utilisateurs autorisés par une stratégie Identity and Access Management (IAM). Lorsque la protection contre la suppression est activée pour une table, personne ne peut la supprimer.

Pour modifier ce paramètre, accédez aux Paramètres supplémentaires de la table, puis au volet Protection contre la suppression et sélectionnez Activer la protection contre la suppression.

La propriété de protection contre la suppression est prise en charge par la console DynamoDB, l'API, CLI/SDK et CloudFormation. L'API `CreateTable` prend en charge la propriété de protection contre la suppression lors de la création de la table, et l'API `UpdateTable` prend en charge la modification de la propriété de protection contre la suppression pour les tables existantes.

> **Note**
>
> - Si un compte AWS est supprimé, toutes les données de ce compte, y compris les tables, sont toujours supprimées dans les 90 jours.
> - Si DynamoDB perd l'accès à une clé gérée par le client ayant permis de chiffrer une table, il archive tout de même la table. L'archivage implique la création d'une sauvegarde de la table et la suppression de l'original.

Liste des noms de table

L'opération `ListTables` renvoie les noms des tables DynamoDB pour le compte et la région AWS actuels.

Example

L'exemple d'AWS CLI suivant montre comment répertorier les noms de tables DynamoDB.

```
aws dynamodb list-tables
```

Description des quotas de débit alloué

L'opération `DescribeLimits` renvoie les quotas actuels de capacité de lecture et d'écriture pour le compte et la région AWS actuels.

Example

L'exemple de AWS CLI suivant montre comment décrire les quotas de débit actuellement alloué.

```
aws dynamodb describe-limits
```

La sortie présente les quotas supérieurs des unités de capacité de lecture et d'écriture pour le compte et la région AWS actuels.

Pour en savoir plus sur ces quotas et la procédure à suivre pour demander une augmentation des quotas, consultez Quotas de débit par défaut.

Points à prendre en considération lors du choix d'une classe de tables dans DynamoDB

DynamoDB propose deux classes de tables conçues pour vous aider à optimiser vos coûts. La classe de tables DynamoDB Standard est la classe par défaut. Elle est recommandée pour la grande majorité des charges de travail. La classe de tables DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) est optimisée pour les tables où le stockage est le coût dominant. Par exemple, les tables qui stockent des données rarement consultées, comme les journaux d'applications, les anciennes publications sur les réseaux sociaux, l'historique des commandes d'e-commerce et les exploits de jeux passés, sont de bons candidats pour la classe de tables Standard-IA.

Chaque table DynamoDB est associée à une classe de tables. Tous les index secondaires associés à la table utilisent la même classe de table. Vous pouvez définir votre classe de tables lors de la création de votre table (DynamoDB Standard par défaut) et mettre à jour la classe de tables d'une table existante à l'aide de AWS Management Console, de AWS CLI, ou du kit SDK AWS. DynamoDB prend également en charge la gestion de votre classe de tables à l'aide de AWS CloudFormation pour les tables à région unique (les tables qui ne sont pas des tables globales). Chaque classe de tables offre une tarification différente pour le stockage de données, ainsi que pour les demandes de lecture et d'écriture. Lorsque vous choisissez une classe de tables pour votre table, gardez à l'esprit les points suivants :

- La classe de tables DynamoDB Standard offre des coûts de débit inférieurs à ceux de DynamoDB Standard-IA et constitue l'option la plus économique pour les tables dont le débit est le coût dominant.
- La classe de tables DynamoDB Standard-IA offre des coûts de stockage inférieurs à ceux de DynamoDB Standard et constitue l'option la plus économique pour les tables dont le stockage est le coût dominant. Lorsque le stockage dépasse 50 % du coût de débit (lectures et écritures) d'une table utilisant la classe de tables DynamoDB Standard, la classe de tables DynamoDB Standard-IA peut vous aider à réduire le coût total de votre table.
- Les tables DynamoDB Standard-IA offrent les mêmes performances, durabilité et disponibilité que les tables DynamoDB Standard.

- Basculer entre les classes de tables DynamoDB Standard et DynamoDB Standard-IA ne nécessite pas de modification du code de votre application. Vous utilisez les mêmes points de terminaison de service et API DynamoDB, quelle que soit la classe de tables utilisée par vos tables.
- Les tables DynamoDB Standard-IA sont compatibles avec toutes les fonctionnalités DynamoDB existantes, comme l'autoscaling, le mode à la demande, le time-to-live (TTL), les sauvegardes à la demande, la reprise ponctuelle (PITR) et les index secondaires globaux.

La classe de tables la plus économique pour votre table dépend des modèles de stockage et d'utilisation du débit attendus de votre table. Vous pouvez consulter l'historique du stockage, ainsi que du coût et de l'utilisation du débit de votre table avec AWS Cost and Usage Reports et AWS Cost Explorer. Ces données historiques vous permettent de déterminer la classe de tables la plus économique pour votre table. Pour en savoir plus sur l'utilisation de AWS Cost and Usage Reports et AWS Cost Explorer, consultez la AWSDocumentation sur la facturation et la gestion des coûts. Pour en savoir plus sur la tarification des classes de tables, consultez Tarification Amazon DynamoDB.

> **Note**
>
> Une mise à jour de classe de tables est un processus d'arrière-plan. Lors de la mise à jour d'une classe de tables, vous pouvez toujours accéder normalement à votre table. La durée de mise à jour de votre classe de tables dépend du trafic de votre table, de la taille de stockage et d'autres variables associées. Sur une période de 30 jours, jusqu'à deux mises à jour de classe de tables sont autorisées sur votre table.

Ajout de balises et d'étiquettes aux ressources dans DynamoDB

Vous pouvez libeller des ressources Amazon DynamoDB à l'aide d'étiquettes. Les balises vous permettent de classer vos ressources de différentes manières, par exemple, par objectif, par propriétaire, par environnement ou selon d'autres critères. Les balises vous permettent d'effectuer les actions suivantes :

- Identifier rapidement une ressource en fonction des balises que vous lui avez attribuées.
- Consultez AWS les factures ventilées par tags.

> **Note**
>
> Les index secondaires locaux (LSI) et les index secondaires globaux (GSI) associés aux tables balisées sont étiquetés avec les mêmes balises automatiquement. Actuellement, l'utilisation DynamoDB Streams ne peut pas être étiquetée.

Le balisage est pris en charge par AWS des services tels qu'Amazon EC2, Amazon S3, DynamoDB, etc. Un balisage efficace peut fournir un aperçu sur les coûts en vous permettant de créer des rapports sur les services associés à une balise spécifique.

Pour commencer le balisage, procédez comme suit :

1. Comprendre Restrictions d'étiquetage dans DynamoDB.
2. Créer des balises en utilisant Étiquetage de ressources dans DynamoDB.
3. Utilisation des balises DynamoDB pour créer des rapports de répartition des coûtsÀ utiliser pour suivre vos AWS coûts par tag actif.

En dernier lieu, il est conseillé de suivre les politique de balisage optimales. Pour plus d'informations, consultez Politiques d'étiquetage AWS.

Restrictions d'étiquetage dans DynamoDB

Chaque étiquette est constituée d'une clé et d'une valeur, que vous définissez. Les restrictions suivantes s'appliquent :

- Chaque table DynamoDB ne peut avoir qu'une seule étiquette avec la même clé. Si vous tentez d'ajouter une balise existante (même clé), la valeur de la balise existante est mise à jour avec la nouvelle valeur.
- Les clés et les valeurs des balises distinguent les majuscules et minuscules.
- La longueur de clé maximale est de 128 caractères Unicode.
- La longueur de valeur maximale est de 256 caractères Unicode.
- Les caractères autorisés sont les lettres, les espaces blancs et les chiffres, ainsi que les caractères spéciaux suivants : `+ - = . _ : /`
- Le nombre maximum d'identifications par ressource est de 50.
- La taille maximale prise en charge pour toutes les balises d'une table est de 10 Ko.

- AWS-les noms et valeurs des balises assignés reçoivent automatiquement le `aws:` préfixe, que vous ne pouvez pas attribuer.AWS-les noms de balises attribués ne sont pas pris en compte dans la limite de 50 balises ni dans la limite de taille maximale de 10 Ko. Les noms des balises attribuées par l'utilisateur ont le préfixe `user:` dans le rapport de répartition des coûts.
- Vous ne pouvez pas antidater l'application d'une balise.

Étiquetage de ressources dans DynamoDB

Vous pouvez utiliser la console Amazon DynamoDB ou AWS Command Line Interface le AWS CLI() pour ajouter, répertorier, modifier ou supprimer des balises. Vous pouvez ensuite activer ces balises définies par l'utilisateur afin de les faire apparaître sur la console AWS Billing and Cost Management pour le suivi de la répartition des coûts. Pour de plus amples informations, veuillez consulter Utilisation des balises DynamoDB pour créer des rapports de répartition des coûts.

Pour la modification en bloc, vous pouvez également utiliser l'éditeur d'étiquettes sur l'AWS Management Console. Pour plus d'informations, consultez Utilisation de Tag Editor.

Pour utiliser l'API DynamoDB à la place, découvrez les opérations suivantes dans la Référence d'API Amazon DynamoDB :

- TagResource
- UntagResource
- ListTagsOfResource

Rubriques

- Définition d'autorisations pour filtrer par étiquettes
- Ajout d'étiquettes à des tables nouvelles ou existantes (AWS Management Console)
- Ajout d'étiquettes à des tables nouvelles ou existantes (AWS CLI)

Définition d'autorisations pour filtrer par étiquettes

Pour utiliser des étiquettes afin de filtrer votre liste de tables dans la console DynamoDB, assurez-vous que les politiques de votre utilisateur incluent l'accès aux opérations suivantes :

- `tag:GetTagKeys`
- `tag:GetTagValues`

Pour accéder à ces opérations en attachant une nouvelle politique IAM à votre utilisateur, suivez les étapes ci-dessous.

1. Accédez à la console IAM avec un utilisateur administrateur.
2. Dans le menu de navigation à gauche, sélectionnez « Policies (politiques) ».
3. Sélectionnez « Create Policy (Créer une politique) ».
4. Collez le politique suivante dans l'éditeur JSON.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Suivez la procédure de l'Assistant et attribuez un nom à la politique (par exemple, `TagKeysAndValuesReadAccess`).
6. Dans le menu de navigation de gauche, sélectionnez « Users (Utilisateurs) ».
7. Dans la liste, sélectionnez l'utilisateur que vous utilisez normalement pour accéder à la console DynamoDB.
8. Sélectionnez « Add permissions (Ajouter des autorisations) ».
9. Sélectionnez « Attach existing policies directly (Attacher directement des politiques existantes) ».
10. Dans la liste, sélectionnez la politique que vous avez créée précédemment.
11. Exécutez l'assistant.

Ajout d’étiquettes à des tables nouvelles ou existantes (AWS Management Console)

Vous pouvez utiliser la console DynamoDB pour ajouter des étiquettes à de nouvelles tables lors de leur création, ou ajouter, modifier ou supprimer des étiquettes pour des tables existantes.

Pour baliser des ressources lors de leur création (console)

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation, choisissez Tables, puis Create table (Créer une table).
3. Sur la page Create DynamoDB table (Créer une table DynamoDB), fournissez un nom et une clé primaire. Dans Identifications, choisissez Ajouter une nouvelle identification et saisissez les identifications que vous souhaitez utiliser.

   Pour plus d’informations sur la structure des balises, consultez Restrictions d’étiquetage dans DynamoDB.

   Pour plus d’informations sur la création de tables, consultez Opérations de base sur les tables DynamoDB.

Pour baliser des ressources existantes (console)

Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/

1. Dans le volet de navigation, choisissez Tables.
2. Choisissez une table dans la liste, puis choisissez l’onglet Paramètres supplémentaires. Vous pouvez ajouter, modifier ou supprimer vos identifications dans la section Identifications en bas de la page.

Ajout d'étiquettes à des tables nouvelles ou existantes (AWS CLI)

Les exemples suivants montrent comment utiliser le AWS CLI pour spécifier des balises lorsque vous créez des tables et des index, et pour étiqueter des ressources existantes.

Pour baliser des ressources lors de leur création (AWS CLI)

- L’exemple suivant crée une nouvelle table `Movies` et ajoute la balise `Owner` avec la valeur `blueTeam` :

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Pour baliser des ressources existantes (AWS CLI)

- L'exemple suivant ajoute la table `Owner` avec la valeur `blueTeam` pour la table `Movies` :

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Pour répertorier toutes les balises pour une table (AWS CLI)

- L'exemple suivant répertorie toutes les balises associées à la table `Movies` :

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Utilisation des balises DynamoDB pour créer des rapports de répartition des coûts

AWS utilise des balises pour organiser les coûts des ressources dans votre rapport de répartition des coûts.AWS fournit deux types de balises de répartition des coûts :

- Une balise AWS générée.AWS définit, crée et applique cette balise pour vous.
- Etiquettes définies par l'utilisateur Vous définissez, créez et appliquez ces étiquettes.

Vous devez activer les deux types d'étiquettes séparément pour qu'elles apparaissent dans Cost Explorer ou sur un rapport de répartition des coûts.

Pour activer les balises AWS générées :

1. Connectez-vous à la console Billing AWS Management Console and Cost Management et ouvrez-la à l'adresse https://console.aws.amazon.com/billing/home#/.
2. Dans le volet de navigation, choisissez Balises de répartition des coûts.
3. Sous Identifications de répartition des coûts générées par AWS, choisissez Activer.

Pour activer les étiquettes définies par l'utilisateur :

1. Connectez-vous à la console Billing AWS Management Console and Cost Management et ouvrez-la à l'adresse https://console.aws.amazon.com/billing/home#/.
2. Dans le volet de navigation, choisissez Balises de répartition des coûts.
3. Sous User-Defines Cost Allocation Tags (Etiquettes de répartition des coûts définies par l'utilisateur), choisissez Activer.

Après avoir créé et activé les balises,AWS génère un rapport de répartition des coûts avec votre utilisation et vos coûts regroupés en fonction de vos balises actives. Le rapport de répartition des coûts inclut tous vos AWS coûts pour chaque période de facturation. Ce rapport inclut les ressources balisées et non balisées, afin que vous puissiez organiser clairement les frais pour les ressources.

> **Note**
>
> Actuellement, les données transférées à partir de DynamoDB ne sont pas réparties par étiquettes sur les rapports de répartition des coûts.

Pour plus d'informations, consultez Utilisation des étiquettes d'allocation des coûts.

Tables globales : réplication multi-active, multirégion

Les tables globales d'Amazon DynamoDB sont une fonctionnalité de base de données entièrement gérée, à plusieurs régions et à multiples activités, qui permet une réplication facile des données et offre des performances de lecture et d'écriture élevées pour les applications mises à l'échelle de manière globale.

Les tables globales répliquent automatiquement les données de vos tables DynamoDB sans que vous ayez Régions AWSà créer et à gérer votre propre solution de réplication. Lorsque vous choisissez les Régions AWS pour les réplicas de vos tables, les tables globales gèrent

automatiquement toutes les réplications. Les tables globales sont disponibles dans toutes les régions où DynamoDB est disponible.

Les tables globales offrent les avantages suivants :

- Répliquez automatiquement les données des tables DynamoDB dans les régions de votre choix pour localiser les données au plus près AWS de vos utilisateurs
- Améliorez la disponibilité des applications en cas d'isolation ou de dégradation au niveau régional
- Éliminez la résolution des conflits de mise à jour afin de pouvoir vous concentrer sur la logique métier de votre application

Les tables globales sont idéales pour les applications nécessitant une continuité des activités et une haute disponibilité grâce à un déploiement multirégional. Tout réplica de table globale peut être utilisé en lecture et en écriture. Les applications peuvent atteindre une résilience élevée avec un objectif de point de reprise (RPO) faible ou nul en transférant le trafic vers une autre région si le traitement des demandes est interrompu dans une région.

Vous pouvez configurer une table globale à l'aide de la console AWS de gestion. Les tables globales utilisent DynamoDB APIs existant pour lire et écrire des données dans vos tables. Aucune modification de l'application n'est donc requise. Vous payez uniquement les ressources provisionnées sans frais à l'avance ni engagement. Pour une mise en route avec les tables globales, consultez Tutoriels : Création de tables globales.

Rubriques

- Fonctionnement des tables globales DynamoDB
- Tutoriels : Création de tables globales
- Sécurité des tables globales DynamoDB
- Présentation de la facturation Amazon DynamoDB pour les tables globales
- Versions des tables globales DynamoDB
- Bonnes pratiques relatives aux tables globales

Fonctionnement des tables globales DynamoDB

Les sections suivantes décrivent les concepts et les comportements des tables globales dans Amazon DynamoDB.

Concepts

Les tables globales sont une fonctionnalité de DynamoDB qui réplique les données des tables entre les régions. AWS

Une table de réplica (ou réplica) est une table DynamoDB unique qui fonctionne comme une partie d'une table globale. Une table globale se compose d'au moins deux répliques de tables réparties dans différentes AWS régions. Une table globale donnée ne peut avoir qu'une seule table de réplica par région AWS . Toutes les répliques d'une table globale partagent le même nom de table, le même schéma de clé primaire et les mêmes données d'élément.

Quand une application écrit des données dans une réplique d'une région, DynamoDB réplique automatiquement l'écriture aux autres réplicas dans la table globale. Pour en savoir plus sur la mise en route avec des tables globales, consultez Tutoriels : Création de tables globales.

Versions

Deux versions des tables globales DynamoDB sont disponibles : la version 2019.11.21 (actuelle) et la version 2017.11.29 (héritée). Vous devez utiliser la version 2019.11.21 (actuelle) dans la mesure du possible. Les informations contenues dans cette section de documentation concernent la version 2019.11.21 (actuelle). Pour de plus amples informations, veuillez consulter Détermination de la version d'une table globale.

Disponibilité

Les tables globales contribuent à améliorer la continuité de votre activité en facilitant la mise en œuvre d'une architecture de haute disponibilité multirégionale. Si la charge de travail d'une seule AWS région est altérée, vous pouvez déplacer le trafic de l'application vers une autre région et effectuer des lectures et des écritures dans une autre table de réplication de la même table globale.

Chaque table de réplica d'une table globale offre la même durabilité et la même disponibilité qu'une table DynamoDB à région unique. Les tables globales offrent un contrat de niveau de service (SLA) avec une disponibilité de 99,999 %, contre 99,99 % pour les tables à région unique.

Modes de cohérence

Lorsque vous créez une table globale, vous pouvez configurer son mode de cohérence. Les tables globales prennent en charge deux modes de cohérence : la cohérence à terme multirégionale (MREC) et la forte cohérence multirégionale (MRSC).

Si vous ne spécifiez pas de mode de cohérence lors de la création d'une table globale, la table globale est définie par défaut sur la cohérence à terme multirégionale (MREC). Une table globale ne peut pas contenir de réplicas configurés avec différents modes de cohérence. Vous ne pouvez pas modifier le mode de cohérence d'une table globale après sa création.

Cohérence à terme multirégionale (MREC)

La cohérence à terme multirégionale (MREC) est le mode de cohérence par défaut pour les tables globales. Les modifications d'éléments dans une réplique de table globale MREC sont répliquées de manière asynchrone sur toutes les autres répliques, généralement en une seconde ou moins. Dans le cas peu probable où une réplique d'une table globale MREC serait isolée ou altérée, toutes les données non encore répliquées dans d'autres régions seront répliquées lorsque la réplique sera saine.

Si le même élément est modifié simultanément dans plusieurs régions, DynamoDB résoudra le conflit en utilisant la modification avec le dernier horodatage interne sur une base par élément, ce que l'on appelle la méthode de résolution des conflits « victoire du dernier auteur ». Un élément finira par converger dans toutes les répliques vers la version créée lors de la dernière écriture.

Les opérations de lecture fortement cohérentes renvoient la dernière version d'un élément si cet élément a été mis à jour pour la dernière fois dans la région où la lecture a eu lieu, mais peuvent renvoyer des données périmées si l'article a été mis à jour pour la dernière fois dans une autre région. Les écritures conditionnelles évaluent l'expression de condition par rapport à la version de l'élément dans la région.

Vous créez une table globale MREC en ajoutant une réplique à une table DynamoDB existante. L'ajout d'un réplica n'a aucun impact sur les performances des tables DynamoDB à région unique ou des réplicas de tables globales existantes. Vous pouvez ajouter des réplicas à une table globale MREC pour augmenter le nombre de régions dans lesquelles les données sont répliquées, ou supprimer des réplicas d'une table globale MREC si elles ne sont plus nécessaires. Une table globale MREC peut comporter un réplica dans n'importe quelle région dans laquelle DynamoDB est disponible, et elle peut comporter autant de réplicas que de régions dans la partition AWS.

Forte cohérence multirégionale (MRSC)

Vous pouvez configurer le mode MRSC (forte cohérence multirégionale) lorsque vous créez une table globale. Les modifications d'éléments dans un réplica de table globale MRSC sont répliquées de manière synchrone dans au moins une autre région avant que l'opération d'écriture ne renvoie une réponse réussie. Lorsque vous convertissez une table à région unique existante en table globale

MRSC, vous devez vous assurer que la table est vide jusqu'à ce que la conversion soit terminée afin de garantir une initialisation et une configuration de réplication correctes.

Les opérations de lecture fortement cohérente sur tout réplica MRSC renvoient toujours la dernière version d'un élément. Les écritures conditionnelles évaluent toujours l'expression de condition par rapport à la dernière version d'un élément.

Une opération d'écriture échoue avec une `ReplicatedWriteConflictException` lorsqu'elle tente de modifier un élément déjà en cours de modification dans une autre région. Les écritures qui échouent avec le `ReplicatedWriteConflictException` peuvent être relancées. Elles réussissent si l'élément n'est plus modifié dans une autre région.

Vous pouvez configurer une table globale MRSC avec trois réplicas, ou avec deux réplicas et un témoin. Un témoin est un composant d'une table globale MRSC qui contient des données écrites dans des réplicas de tables globales. Il fournit une alternative facultative à un réplica complet tout en prenant en charge l'architecture de disponibilité de la MRSC. Vous ne pouvez pas effectuer d'opérations de lecture ou d'écriture sur un témoin. Un témoin se trouve dans une région différente de celle des deux réplicas.

Lorsque vous créez une table globale MRSC, vous choisissez les régions pour vos répliques et pour le déploiement témoin au moment de la création de la table MRSC. Vous pouvez déterminer si et dans quelle région une table globale MRSC comporte un témoin configuré à partir de la sortie de l'API `DescribeTable`. Le témoin est détenu et géré par DynamoDB, et il n'apparaîtra pas dans AWS votre compte dans la région où il est configuré.

Une table globale MRSC doit être déployée dans exactement trois régions. Vous créez une table globale MRSC en ajoutant un réplica et un témoin, ou deux réplicas à une table DynamoDB existante qui ne contient aucune donnée. Vous ne pouvez pas ajouter de réplicas supplémentaires à une table globale MRSC existante. Vous ne pouvez pas supprimer un seul réplica ou un seul témoin d'une table globale MRSC. Vous pouvez supprimer deux réplicas ou supprimer un réplica et un témoin d'une table globale MRSC, en convertissant le réplica restant en une table DynamoDB à région unique.

Les considérations suivantes s'appliquent aux tables globales MRSC :

- Lorsque vous convertissez une table à région unique en table globale MRSC, vous devez vous assurer que la table est vide. La conversion d'une table à région unique en une table globale MRSC contenant des éléments existants n'est pas prise en charge. Assurez-vous qu'aucune donnée n'est écrite dans la table pendant le processus de conversion.

- Les tables globales MRSC sont disponibles dans les ensembles de régions suivants :
  - Groupe de régions États-Unis : USA Est (Virginie du Nord), USA Est (Ohio), USA Ouest (Oregon)
  - Groupe de régions UE : Europe (Francfort), Europe (Irlande), Europe (Londres), Europe (Paris)
  - Groupe de régions AP : Asie-Pacifique (Tokyo), Asie-Pacifique (Séoul) et Asie-Pacifique (Osaka).
- Les tables globales MRSC ne peuvent pas couvrir des ensembles régionaux (par exemple, une table globale MRSC ne peut pas contenir de réplicas d'ensembles régionaux des États-Unis et de l'UE).
- La durée de vie (TTL) n'est pas prise en charge pour les tables globales MRSC.
- Les index secondaires locaux (LSIs) ne sont pas pris en charge pour les tables globales MRSC.
- CloudWatch Les informations de Contributor Insights ne sont communiquées que pour la région dans laquelle une opération a eu lieu.

Choix d'un mode de cohérence

Le critère clé pour choisir un mode de cohérence multirégional est de savoir si votre application privilégie les écritures à faible latence et les lectures fortement cohérentes, ou privilégie la cohérence globale.

Les tables globales MREC auront des temps d'écriture plus faibles et des latences de lecture fortement cohérentes par rapport aux tables globales MRSC. Les tables globales MREC prennent en charge un objectif de point de reprise (RPO) égal au délai de réplication entre les réplicas, généralement de quelques secondes selon les régions du réplica.

Vous devez utiliser le mode MREC lorsque :

- Votre application peut tolérer des données obsolètes renvoyées par des opérations de lecture fortement cohérentes si ces données ont été mises à jour dans une autre région.
- Vous privilégiez une écriture plus faible et des latences de lecture fortement cohérentes par rapport à la cohérence de lecture multirégionale.
- Votre stratégie de haute disponibilité multirégionale peut tolérer un RPO supérieur à zéro.

Les tables globales MRSC auront des temps d'écriture plus élevés et des latences de lecture fortement cohérentes par rapport aux tables globales MREC. Les tables MRSC prennent en charge un objectif de point de reprise (RPO) de zéro.

Vous devez utiliser le mode MREC lorsque :

- Vous avez besoin de lectures fortement cohérentes dans plusieurs régions.
- Vous privilégiez la cohérence globale de la lecture plutôt que la réduction de la latence d'écriture.
- Votre stratégie de haute disponibilité multirégionale nécessite un RPO de zéro.

Surveillance des tables globales

Les tables globales configurées pour la cohérence finale multirégionale (MREC) publient la `ReplicationLatency`métrique dans. CloudWatch Cette métrique suit le temps écoulé entre le moment où un élément est écrit dans une table de réplicas et celui où il apparaît dans un autre réplica dans la table globale. `ReplicationLatency` est exprimé en millisecondes et est émis pour chaque paire région source/région de destination dans une table globale.

Les `ReplicationLatency` valeurs typiques dépendent de la distance entre les AWS régions que vous avez choisies, ainsi que d'autres variables telles que le type de charge de travail et le débit. Par exemple, un réplica source située dans la région USA Ouest (Californie du Nord) (us-west-1) a une `ReplicationLatency` inférieure dans la région USA Ouest (Oregon) (us-west-2) par rapport à la région Afrique (Le Cap) (af-south-1).

Une valeur croissante pour `ReplicationLatency` peut indiquer que les mises à jour d'un réplica ne sont pas propagées vers d'autres tables de réplica dans un délai raisonnable. Dans ce cas, vous pouvez rediriger temporairement les activités de lecture et d'écriture de votre application vers une autre AWS région.

Les tables globales configurées pour une forte cohérence multirégionale (MRSC) ne publient aucune métrique `ReplicationLatency`.

Test d'injection de pannes

Les tables globales MREC s'intègrent au service d'injection de AWS défauts (AWS FIS) pour effectuer des expériences d'injection de défauts sur vos charges de travail de tables globales. Cela vous permet de tester la réponse de votre application à une simulation d'isolation de région en interrompant la réplication vers et depuis un réplica sélectionné. Pour plus d'informations, consultez la mise en pause de la réplication d'une table globale.

Durée de vie (TTL)

Les tables globales configurées pour le MREC prennent en charge la configuration de la suppression Time To Live (TTL). Les paramètres TTL sont automatiquement synchronisés pour tous les réplicas d'une table globale. Lorsque la TTL supprime un élément d'un réplica dans une région, la suppression est répliquée sur tous les autres réplicas de la table globale. La TTL n'utilise pas de capacité d'écriture. La suppression TTL ne vous est donc pas facturée dans la région où la suppression a eu lieu. Toutefois, vous êtes facturé pour la suppression répliquée dans chaque autre région avec un réplica dans la table globale.

La réplication de suppression TTL utilise de la capacité d'écriture sur les réplicas dans lesquels la suppression est répliquée. Les réplicas configurés pour la capacité allouée peuvent limiter les demandes si la combinaison du débit d'écriture et du débit de suppression TTL est supérieure à la capacité d'écriture allouée.

Les tables globales configurées pour une forte cohérence multirégionale (MRSC) ne prennent pas en charge la configuration de la suppression Time To Live (TTL).

Flux

Les tables globales configurées pour une cohérence à terme entre plusieurs régions (MREC) répliquent les modifications en lisant ces modifications à partir de flux DynamoDB sur une table de réplica et en appliquant cette modification à toutes les autres tables de réplica. Les flux sont donc activés par défaut sur tous les réplicas d'une table globale MREC et ne peuvent pas être désactivé sur ces réplicas. Le processus de réplication MREC peut combiner plusieurs modifications sur une courte période en une seule écriture répliquée, ce qui fait que le flux de chaque réplica contient des enregistrements légèrement différents. Les enregistrements de flux sur les réplicas MREC sont toujours classés par article, mais l'ordre entre les éléments peut différer selon les réplicas.

Les tables globales configurées pour une forte cohérence multirégionale (tables MRSC) n'utilisent pas les flux DynamoDB pour la réplication. Cette fonctionnalité n'est donc pas activée par défaut sur les réplicas MRSC. Vous pouvez activer les flux sur un réplica MRSC. Les enregistrements de flux sur les répliques MRSC sont identiques pour chaque réplica, y compris l'ordre des enregistrements de flux.

Si vous souhaitez écrire une application qui traite les enregistrements de flux pour les modifications survenues dans une région particulière mais pas dans d'autres régions d'une table globale, vous pouvez ajouter un attribut à chaque élément qui définit dans quelle région le changement pour cet élément s'est produit. Vous pouvez utiliser cet attribut pour filtrer les enregistrements de flux

en fonction des modifications survenues dans d'autres régions, notamment en utilisant des filtres d'événements Lambda pour n'appeler les fonctions Lambda que pour les modifications dans une région spécifique.

Transactions

Sur une table globale configurée pour MREC, les opérations de transaction DynamoDB (`TransactWriteItems` et `TransactGetItems`) ne sont atomiques que dans la région où l'opération a été invoquée. Les écritures transactionnelles ne sont pas répliquées en tant qu'unité entre les régions, ce qui signifie que seules certaines des écritures d'une transaction peuvent être renvoyées par des opérations de lecture dans d'autres réplicas à un instant dans le passé donné.

Par exemple, si vous avez une table globale avec des réplicas dans les régions USA Est (Ohio) et USA Ouest (Oregon), et que vous réalisez une opération `TransactWriteItems` dans la région USA Est (Ohio), vous remarquerez peut-être des transactions partiellement incomplètes dans la région USA Ouest (Oregon) lorsque les changements sont répliqués. Les changements seront uniquement répliqués aux autres régions une fois validés dans la région source.

Les tables globales configurées pour une forte cohérence multirégionale (MRSC) ne prennent pas en charge les opérations de transaction et renvoient une erreur si ces opérations sont invoquées sur une réplique MRSC.

Débit de lecture et d'écriture

La réplication consomme de la capacité d'écriture. Les réplicas configurés pour la capacité allouée peuvent limiter les demandes si la combinaison du débit d'écriture et du débit de réplication est supérieure à la capacité d'écriture allouée. La capacité d'écriture en mode à la demande est synchronisée pour tous les réplicas d'une table globale. Les tables globales configurées pour la capacité provisionnée synchronisent les paramètres d'autoscaling entre les réplicas. Le paramètre de capacité d'écriture réellement provisionnée peut varier entre les réplicas en fonction du débit d'écriture utilisé.

Vous pouvez configurer indépendamment les paramètres de capacité de lecture pour chaque réplica dans une table globale. Lors de l'ajout d'un réplica à une table globale, la capacité de lecture de la table source ou du réplica est utilisée comme valeur initiale, sauf si une valeur de remplacement est spécifiée.

Synchronisation des paramètres

Les paramètres des tables globales DynamoDB sont des paramètres de configuration qui contrôlent divers aspects du comportement des tables et de la réplication. Ces paramètres sont gérés via le APIs plan de contrôle DynamoDB et peuvent être configurés lors de la création ou de la modification de tables globales. Les tables globales synchronisent automatiquement certains paramètres entre tous les réplicas afin de maintenir la cohérence, tout en offrant une certaine flexibilité pour les optimisations spécifiques aux régions. Comprendre quels paramètres sont synchronisés et comment ils se comportent vous permet de configurer efficacement votre table globale. Les paramètres se répartissent en trois catégories principales en fonction de la façon dont ils sont synchronisés entre les réplicas.

Les paramètres suivants sont toujours synchronisés entre les réplicas d'une table globale :

- Mode capacité (capacité provisionnée ou à la demande)
- Capacité d'écriture provisionnée dans la table
- Autoscaling de l'écriture dans une table
- Définition de l'index secondaire global (GSI)
- Capacité d'écriture provisionnée dans GSI
- Autoscaling de l'écriture dans GSI
- Type de chiffrement côté serveur (SSE)
- Définition des flux en mode MREC
- Durée de vie (TTL)
- Débit chaud
- Débit d'écriture maximal à la demande

Les paramètres suivants sont synchronisés entre les réplicas, mais peuvent être remplacés au cas par cas :

- Capacité de lecture provisionnée pour une table
- Autoscaling de la lecture dans une table
- Capacité de lecture provisionnée dans GSI
- Autoscaling de la lecture dans GSI
- Classe de table

- Débit de lecture maximum à la demande

> **Note**
>
> Les valeurs de paramètre remplaçables sont modifiées si le paramètre est modifié sur un autre réplica. Par exemple, vous avez une table globale MREC avec des réplicas dans les régions USA Est (Virginie du Nord) et USA Ouest (Oregon). La réplique de l'est des États-Unis (Virginie du Nord) a prévu un débit de lecture fixé à 200. RCUs La réplique située dans l'ouest des États-Unis (Oregon) dispose d'une dérogation au débit de lecture provisionnée fixée à 100. RCUs Si vous mettez à jour le paramètre de débit de lecture provisionné sur la réplique de l'est des États-Unis (Virginie du Nord) de 200 RCUs à 300 RCUs, la nouvelle valeur du débit de lecture provisionné sera également appliquée à la réplique de l'ouest des États-Unis (Oregon). Cela fait passer le paramètre de débit de lecture provisionné pour la réplique de l'ouest des États-Unis (Oregon) de la valeur remplacée de 100 RCUs à la nouvelle valeur de 300. RCUs

Les paramètres suivants ne sont jamais synchronisés entre les réplicas :

- Protection contre la suppression
- Point-in-time Récupération
- Étiquettes
- Activation de Tableau CloudWatch Contributor Insights
- Activation de GSI CloudWatch Contributor Insights
- Définition de flux de données Kinesis
- Stratégies basées sur une ressource
- Définition des flux en mode MRSC

Tous les autres paramètres ne sont pas synchronisés entre les réplicas.

DynamoDB Accelerator (DAX)

Les écritures dans des réplicas de tables globales contournent DynamoDB Accelerator (DAX) et mettent directement DynamoDB à jour. Par conséquent, les caches DAX peuvent devenir obsolètes, car les écritures ne mettent pas à jour le cache DAX. Les caches DAX configurés pour les réplicas de tables globales ne seront actualisés que lorsque la TTL du cache expirera.

Considérations relatives à la gestion des tables globales

Vous ne pouvez pas supprimer une table utilisée pour ajouter un nouveau réplica de table globale avant que 24 heures ne se soient écoulées depuis la création du nouveau réplica.

Si vous désactivez une AWS région contenant des répliques de tables globales, ces répliques sont définitivement converties en tables à région unique 20 heures après la désactivation de la région.

Tutoriels : Création de tables globales

Cette section fournit des step-by-step instructions pour créer des tables globales DynamoDB configurées pour votre mode de cohérence préféré. Choisissez le mode MREC ou le mode MRSC en fonction des exigences de votre application.

Les tables globales MREC offrent une latence d'écriture plus faible avec une cohérence à terme entre les Régions AWS. Les tables globales MRSC permettent des lectures fortement cohérentes dans toutes les régions avec des latences d'écriture légèrement supérieures à celles des tables MREC. Choisissez le mode de cohérence qui répond le mieux aux besoins de cohérence, de latence et de disponibilité des données de votre application.

Rubriques

- Création d'une table globale configurée pour MREC
- Création d'une table globale configurée pour MRSC

Création d'une table globale configurée pour MREC

Cette section explique comment créer une table globale avec le mode MREC. MREC est le mode de cohérence par défaut pour les tables globales et il permet des écritures à faible latence avec une réplication asynchrone entre les Régions AWS. Les modifications apportées à un élément dans une région sont généralement répliquées dans toutes les autres régions en une seconde. Le mode MREC est donc idéal pour les applications qui privilégient une faible latence d'écriture et peuvent tolérer de brèves périodes pendant lesquelles différentes régions peuvent renvoyer des versions de données légèrement différentes.

Vous pouvez créer des tables globales MREC avec des répliques dans toutes les AWS régions où DynamoDB est disponible et ajouter ou supprimer des répliques à tout moment. Les exemples suivants montrent comment créer une table globale MREC avec des réplicas dans plusieurs régions.

Création d’une table globale MREC à l’aide de la console DynamoDB

Suivez ces étapes pour créer une table globale avec AWS Management Console. L’exemple suivant crée une table globale avec des tables de réplica aux États-Unis et en Europe.

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Pour cet exemple, choisissez USA Est (Ohio) dans le sélecteur de région dans la barre de navigation.
3. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
4. Choisissez Créer une table.
5. Sur la page Créer un tableau :
   a. Sous Nom du tableau, saisissez **`Music`**.
   b. Pour Clé de partition, saisissez **`Artist`**.
   c. Pour Clé de tri, entrez **`SongTitle`**.
   d. Conservez les autres paramètres par défaut et choisissez Créer une table.

      La nouvelle table sert de première table de réplica dans une nouvelle table globale. Il s’agit du prototype pour d’autres tables de réplica que vous ajouterez ultérieurement.
6. Une fois que la table est active :
   a. Choisissez la table Musique dans la liste de tables.
   b. Choisissez l’onglet Tables globales.
   c. Choisissez Créer un réplica.
7. Dans la liste déroulante Régions de réplication disponibles, choisissez USA Ouest (Oregon) us-west-2.

   La console vérifie qu’il n’existe pas de table du même nom dans la région sélectionnée. (Si une table du même nom existe, vous devez la supprimer avant de pouvoir créer une nouvelle table de réplicas dans cette région.)
8. Choisissez Créer un réplica. Cela a pour effet de démarrer le processus de création de table dans la région USA Ouest (Oregon) us-west-2 Region.

   L’onglet Tables globales pour la table Musique (et pour toutes les autres tables de réplica) indique que la table a été répliquée dans plusieurs régions.

9. Ajoutez une autre région en répétant les étapes précédentes, mais choisissez Europe (Francfort) eu-central-1 comme région.
10. Pour tester la réplication :
    a. Assurez-vous d'utiliser le AWS Management Console dans la région de l'est des États-Unis (Ohio).
    b. Choisissez Explorer les éléments de la table.
    c. Choisissez Créer un élément.
    d. Entrez **`item_1`** pour Artiste et **`Song Value 1`** pour SongTitle.
    e. Choisissez Créer un élément.
11. Vérifiez la réplication en passant aux autres régions :
    a. Dans le sélecteur de région situé en haut à droite, choisissez Europe (Francfort).
    b. Vérifiez que le tableau Musique contient l’élément que vous avez créé.
    c. Répétez la vérification pour la région USA Ouest (Oregon).

Création d'une table globale MREC à l'aide de AWS CLI ou Java

CLI

L’exemple de code suivant montre comment gérer les tables globales DynamoDB avec la cohérence à terme de la réplication multirégionale (MREC).

- Créez une table avec la réplication multirégionale (MREC).
- Placez et récupérez des objets de tables de réplica.
- Supprimez les répliques one-by-one.
- Nettoyez en supprimant la table.

AWS CLI avec le script Bash

Créez une table avec la réplication multirégionale.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
```

```
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

Décrivez la table multirégionale.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
```

```
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Placez les objets dans une table de réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Récupérez des objets de tables de réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Supprimez les réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
```

```
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Nettoyez en supprimant la table.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Java

L’exemple de code suivant montre comment créer et gérer des tables globales DynamoDB avec des réplicas entre plusieurs régions.

- Création d’une table avec un index secondaire global et DynamoDB Streams.
- Ajout de réplicas dans différentes régions pour créer une table globale.
- Suppression de réplicas d’une table globale.
- Ajout d’éléments de test pour vérifier la réplication entre plusieurs régions.
- Description de la configuration globale de la table et de l’état du réplica.

SDK pour Java 2.x

Créez une table avec l'index secondaire global et les flux DynamoDB à l'aide de.AWS SDK for Java 2.x

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
```

```
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
```

```
    }
```

Attendez qu'une table soit active en utilisant AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Ajoutez une réplique pour créer ou étendre une table globale à l'aide de AWS SDK for Java 2.x.

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
```

```
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();
```

```
            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Supprimez une réplique d'une table globale à l'aide de AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();
```

```
            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Ajoutez des éléments de test pour vérifier la réplication à l'aide de AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
```

```
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Décrivez la configuration globale des tables et les répliques à l'aide AWS SDK for Java 2.x de.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Exemple complet d'opérations de table globales utilisant AWS SDK for Java 2.x.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {
```

```
        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
```

```
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK for Java 2.x.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Création d'une table globale configurée pour MRSC

Cette section explique comment créer une table globale à forte cohérence multirégionale (MRSC). Les tables globales MRSC reproduisent de manière synchrone les modifications apportées aux éléments d'une région à l'autre, ce qui garantit que les opérations de lecture fortement cohérentes sur tout réplica renvoient toujours la dernière version d'un élément. Lorsque vous convertissez une table à région unique en table globale MRSC, vous devez vous assurer que la table est vide. La conversion d'une table à région unique en une table globale MRSC contenant des éléments existants n'est pas prise en charge. Assurez-vous qu'aucune donnée n'est écrite dans la table pendant le processus de conversion.

Vous pouvez configurer une table globale MRSC avec trois réplicas, ou deux réplicas et un témoin. Lorsque vous créez une table globale MRSC, vous choisissez les régions dans lesquelles les réplicas et un témoin facultatif sont déployés. L'exemple suivant crée une table globale MRSC avec des

réplicas dans les régions USA Est (Virginie du Nord) et USA Est (Ohio) et USA Est (Ohio) et un témoin dans la région USA Ouest (Oregon).

> **Note**
>
> Avant de créer une table globale, vérifiez que les limites de débit des quotas de service sont cohérentes dans toutes les régions cibles, car cela est nécessaire pour créer une table globale. Pour plus d'informations sur les limites de débit globales des tables, consultez Global tables quotas.

Création d'une table globale MRSC à l'aide de la console DynamoDB

Suivez ces étapes pour créer une table globale MRSC avec l'AWS Management Console.

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le sélecteur de région dans la barre de navigation, choisissez une région où les tables globales avec MRSC sont prises en charge, par exemple **`us-east-2`**.
3. Dans le volet de navigation, choisissez Tables.
4. Choisissez Créer un tableau.
5. Sur la page Créer un tableau :
   a. Sous Nom du tableau, saisissez **`Music`**.
   b. Pour Clé de partition, saisissez **`Artist`**, et conservez le type de Chaîne par défaut.
   c. Pour Clé de tri, saisissez **`SongTitle`**, et conservez le type de Chaîne par défaut.
   d. Conservez les autres paramètres par défaut et choisissez Créer une table

      La nouvelle table sert de première table de réplica dans une nouvelle table globale. Il s'agit du prototype pour d'autres tables de réplica que vous ajouterez ultérieurement.
6. Attendez que la table soit active, puis sélectionnez-la dans la liste des tables.
7. Choisissez l'onglet Tables globales, puis choisissez Créer un réplica.
8. Sur la page Créer un réplica :
   a. Sous Cohérence multirégionale, sélectionnez Forte cohérence.
   b. Pour Région de réplication 1, choisissez **`US East (N. Virginia) us-east-1`**.

c. Pour Région de réplication 2, choisissez **US West (Oregon) us-west-2**.

d. Cochez la case Configurer en tant que témoin pour la région USA Ouest (Oregon).

e. Choisissez Créer des réplicas.

9. Attendez que le processus de création du réplica et du témoin soit terminé. L'état du réplica affiché est Actif lorsque la table est prête pour utilisation.

Création d'une table globale MRSC à l'aide de AWS CLI ou Java

Avant de commencer, assurez-vous que votre principal IAM dispose des autorisations requises pour créer une table globale MRSC avec une région témoin.

L'exemple de politique IAM suivant vous permet de créer une table DynamoDB (`MusicTable`) dans la région USA Est (Ohio) avec un réplica dans la région USA Est (Virginie du Nord) et une région témoin dans la région USA Ouest (Oregon) :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
```

```
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-east-2:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicTable"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "replication.dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

Les exemples de code suivants montrent comment créer et gérer des tables globales DynamoDB avec MRSC (forte cohérence multirégionale).

- Création d'une table avec une forte cohérence multirégionale.
- Vérifiez la configuration MRSC et le statut du réplica.
- Testez une forte cohérence entre les régions grâce à des lectures immédiates.
- Effectuez des écritures conditionnelles avec les garanties MRSC.
- Nettoyez les ressources des tables globales MRSC.

Bash

AWS CLI avec le script Bash

Créez une table avec une forte cohérence multirégionale.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
```

```
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Vérifiez la configuration MRSC et le statut du réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Testez une forte cohérence avec des lectures immédiates dans les régions.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
```

```
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Effectuez des écritures conditionnelles avec les garanties MRSC.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Nettoyez les ressources des tables globales MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK pour Java 2.x

Créez un tableau régional prêt pour la conversion MRSC à l'aide AWS SDK for Java 2.x de.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
```

```
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Convertissez une table régionale en MRSC à l'aide de répliques et de témoins à l'aide de témoins.AWS SDK for Java 2.x

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
```

```
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Décrivez une configuration de table globale MRSC à l'aide AWS SDK for Java 2.x de.

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
```

```
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Ajout d’éléments de test pour vérifier la forte cohérence du MRSC à l’aide du kit AWS SDK for Java 2.x.

```
   public static PutItemResponse putTestItem(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final String artist,
       final String songTitle,
       final String album,
       final String year) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Adding test item to MRSC global table: " + tableName);

           Map<String, AttributeValue> item = new HashMap<>();
           item.put("Artist", AttributeValue.builder().s(artist).build());
           item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           if (album != null && !album.trim().isEmpty()) {
               item.put("Album", AttributeValue.builder().s(album).build());
           }
           if (year != null && !year.trim().isEmpty()) {
               item.put("Year", AttributeValue.builder().n(year).build());
           }

           PutItemRequest putItemRequest =
               PutItemRequest.builder().tableName(tableName).item(item).build();

           PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
           LOGGER.info("Test item added successfully with strong consistency");
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lisez des éléments avec des lectures cohérentes à partir de répliques MRSC à l'aide de.AWS SDK for Java 2.x

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());
```

```
            GetItemRequest getItemRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Effectuez des mises à jour conditionnelles avec les garanties MRSC en utilisant AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");
```

```
            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Attendez que les répliques du MRSC et que les témoins deviennent actifs en utilisant.AWS SDK for Java 2.x

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds
```

```
            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
```

```
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }
```

```
            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Nettoyez les répliques du MRSC et les témoins à l'aide de.AWS SDK for Java 2.x

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
```

```
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Démonstration complète du flux de travail MRSC à l'aide de AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
```

```
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);
```

```
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
```

```
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK for Java 2.x.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Sécurité des tables globales DynamoDB

Les répliques de tables globales étant des tables DynamoDB, vous utilisez les mêmes méthodes pour contrôler l'accès aux répliques que pour les tables à région unique,Gestion des identités et des accès AWS y compris les politiques d'identité (IAM) et les politiques basées sur les ressources.

Cette rubrique explique comment sécuriser les tables globales DynamoDB à l'aide des autorisations AWS Key Management Service IAM et du chiffrement ().AWS KMS Vous découvrirez les rôles liés aux services (SLR) qui permettent la réplication entre régions et l'auto-scaling, les autorisations IAM nécessaires pour créer, mettre à jour et supprimer des tables globales, ainsi que les différences entre les tables de cohérence finale multirégionale (MREC) et les tables de cohérence forte multirégions (MRSC). Vous découvrirez également les clés de AWS KMS chiffrement permettant de gérer la réplication entre régions en toute sécurité.

Rôles liés à un service pour les tables globales

Les tables globales DynamoDB s'appuient sur des rôles liés à un service SLRs () pour gérer les fonctionnalités de réplication entre régions et d'auto-scaling.

Vous ne devez configurer ces rôles qu'une seule fois par AWS compte. Une fois créés, les mêmes rôles sont utilisés pour toutes les tables globales de votre compte. Pour en savoir plus sur l'utilisation des rôles liés à un service, consultez Utilisation des rôles liés à un service dans le Guide de l'utilisateur IAM.

Rôle lié à un service pour la réplication

Amazon DynamoDB crée automatiquement `AWSServiceRoleForDynamoDBReplication` le rôle lié à un service (SLR) lorsque vous créez votre première table globale. Ce rôle gère la réplication entre régions pour vous.

Lorsque vous appliquez des politiques basées sur les ressources à des répliques, assurez-vous de ne refuser aucune des autorisations définies dans le `AWSServiceRoleForDynamoDBReplicationPolicy` SLR principal, car cela interrompra la réplication. Si vous refusez les autorisations SLR requises, la réplication vers et depuis les réplicas concernés s'arrêtera et le statut de la table de réplicas passera à `REPLICATION_NOT_AUTHORIZED`.

- Pour les tables globales MREC (Multi-region Evental Cohérence), si une réplique reste dans l'`REPLICATION_NOT_AUTHORIZED`état pendant plus de 20 heures, elle est convertie de manière irréversible en table DynamoDB à région unique.
- Pour les tables globales à forte cohérence multirégionale (MRSC), le refus des autorisations requises entraîne des opérations `AccessDeniedException` d'écriture et de lecture très cohérentes. Si une réplique reste dans `REPLICATION_NOT_AUTHORIZED` cet état pendant plus de sept jours, elle devient définitivement inaccessible, et les opérations d'écriture et de lecture très cohérentes continueront d'échouer avec une erreur. Certaines opérations de gestion, telles que la suppression des réplicas, seront couronnées de succès.

Rôle lié à un service pour l'autoscaling

Lors de la configuration d'une table globale pour le mode capacité allouée, le dimensionnement automatique doit être configuré pour la table globale. DynamoDB Auto Scaling AWS utilise le service Application Auto Scaling pour ajuster dynamiquement la capacité de débit allouée sur vos répliques de tables globales. Le service Application Auto Scaling crée un rôle lié au service (SLR) nommé. `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable` Ce rôle lié à un service est automatiquement créé dans votre AWS compte lorsque vous configurez pour la première fois le dimensionnement automatique pour une table DynamoDB. Il permet à Application Auto Scaling de gérer la capacité des tables provisionnées et de créer des CloudWatch alarmes.

Lorsque vous appliquez des politiques basées sur les ressources à des répliques, assurez-vous de ne pas refuser les autorisations définies dans le `AWSApplicationAutoscalingDynamoDBTablePolicy`principal Application Auto Scaling SLR, car cela interromprait la fonctionnalité de dimensionnement automatique.

Exemples de politiques IAM pour les rôles liés à un service

Une politique IAM présentant la condition suivante n'a aucune incidence sur les autorisations requises pour le SLR de réplication DynamoDB et AWS le SLR Auto Scaling. Cette condition peut être ajoutée à des politiques par ailleurs largement restrictives afin d'éviter d'interrompre involontairement la réplication ou le dimensionnement automatique.

Exclusion des autorisations SLR requises des politiques de refus

L'exemple suivant montre comment exclure les principaux de rôles liés à un service des déclarations de refus :

```
"Condition": {
    "StringNotEquals": {
        "aws:PrincipalArn": [
            "arn:aws::iam::111122223333:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "arn:aws::iam::111122223333:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        ]
    }
}
```

Comment les tables globales utilisent AWS IAM

Les sections suivantes décrivent les autorisations requises pour les différentes opérations sur les tables globales et fournissent des exemples de politiques pour vous aider à configurer l'accès approprié pour vos utilisateurs et applications.

> **Note**
>
> Toutes les autorisations décrites doivent être appliquées à l'ARN de la ressource de table spécifique dans la ou les régions concernées. L'ARN de la ressource de table suit le format `arn:aws:dynamodb:region:account-id:table/table-name` dans lequel vous devez spécifier les valeurs réelles de votre région, de votre identifiant de compte et du nom de la table.

Rubriques

- Création de tables globales et ajout de répliques

- Mise à jour de tables globales
- Supprimer des tables globales et supprimer des répliques

Création de tables globales et ajout de répliques

Les tables globales DynamoDB prennent en charge deux modes de cohérence : cohérence finale multirégionale (MREC) et cohérence forte multirégionale (MRSC). Les tables globales MREC peuvent avoir plusieurs répliques dans un certain nombre de régions et garantir une cohérence finale. Les tables mondiales MRSC nécessitent exactement trois régions (trois répliques ou deux répliques et un témoin) et garantissent une forte cohérence avec un objectif de point de reprise zéro (RPO).

Les autorisations requises pour créer des tables globales varient selon que vous créez une table globale avec ou sans témoin.

Autorisations pour créer des tables globales

Les autorisations suivantes sont requises à la fois pour la création initiale de la table globale et pour l'ajout de répliques ultérieurement. Ces autorisations s'appliquent à la fois aux tables globales Multi-Region Evental Consistency (MREC) et Multi-Region Strong Consistency (MRSC).

- Les tables globales nécessitent une réplication entre régions, que DynamoDB gère via le rôle lié à un service (`AWSServiceRoleForDynamoDBReplication`SLR). L'autorisation suivante permet à DynamoDB de créer ce rôle automatiquement lorsque vous créez une table globale pour la première fois :
  - `iam:CreateServiceLinkedRole`
- Pour créer une table globale ou ajouter une réplique à l'aide de l'`UpdateTable`API, vous devez disposer des autorisations suivantes sur la ressource de la table source :
  - `dynamodb:UpdateTable`
- Vous devez disposer des autorisations suivantes sur la ressource de table dans les régions pour que les répliques soient ajoutées :
  - `dynamodb:CreateTable`
  - `dynamodb:CreateTableReplica`
  - `dynamodb:Query`
  - `dynamodb:Scan`
  - `dynamodb:UpdateItem`
  - `dynamodb:PutItem`

- dynamodb:GetItem
- dynamodb:DeleteItem
- dynamodb:BatchWriteItem

Autorisations supplémentaires pour les tables globales MRSC utilisant un témoin

Lorsque vous créez une table globale multirégionale à forte cohérence (MRSC) avec une région témoin, vous devez disposer des autorisations suivantes sur la ressource de la table dans toutes les régions participantes (y compris les régions répliques et la région témoin) :

- dynamodb:CreateGlobalTableWitness

Exemples de politiques IAM pour créer des tables globales

Création d'une table mondiale MREC ou MRSC dans trois régions

La politique basée sur l'identité suivante vous permet de créer une table globale MREC ou MRSC nommée « utilisateurs » dans trois régions, notamment en créant le rôle lié au service de réplication DynamoDB requis.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
```

```
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Restreindre la création de tables globales MREC ou MRSC à des régions spécifiques

La politique basée sur l'identité suivante vous permet de créer des répliques de tables globales DynamoDB dans des régions spécifiques à l'aide de la clé de RequestedRegion condition aws :, notamment en créant le rôle lié au service de réplication DynamoDB requis.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAddingReplicasToSourceTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
```

```
            "us-east-1"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-2",
            "us-west-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Création d'une table mondiale du MRSC avec témoins

La politique basée sur l'identité suivante vous permet de créer une table globale DynamoDB MRSC nommée « users » avec des répliques dans us-east-1 et us-east-2 et un témoin dans us-west-2, notamment en créant le rôle lié au service de réplication DynamoDB requis.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTableWithWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:CreateGlobalTableWitness",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
```

```
  ]
}
```

Restreindre la création de témoins MRSC à des régions spécifiques

Cette politique basée sur l'identité vous permet de créer une table globale MRSC avec des répliques limitées à des régions spécifiques à l'aide de la clé de RequestedRegion condition aws : et de la création illimitée de témoins dans toutes les régions, y compris la création du rôle lié au service de réplication DynamoDB requis.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-east-2"
          ]
        }
      }
    },
    {
```

```
      "Sid": "AllowCreatingWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateGlobalTableWitness"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Mise à jour de tables globales

Pour modifier les paramètres de réplication d'une table globale existante à l'aide de l'UpdateTableAPI, vous devez disposer de l'autorisation suivante sur la ressource de table dans la région dans laquelle vous effectuez l'appel d'API :

- `dynamodb:UpdateTable`

Vous pouvez également mettre à jour d'autres configurations de table globales, telles que les politiques de dimensionnement automatique et les paramètres Time to Live. Les autorisations suivantes sont requises pour ces opérations de mise à jour supplémentaires :

- Pour mettre à jour une politique de dimensionnement automatique des répliques avec l'UpdateTableReplicaAutoScalingAPI, vous devez disposer des autorisations suivantes sur la ressource de table dans toutes les régions contenant des répliques :
  - `application-autoscaling:DeleteScalingPolicy`
  - `application-autoscaling:DeleteScheduledAction`
  - `application-autoscaling:DeregisterScalableTarget`

  - `application-autoscaling:DescribeScalableTargets`
  - `application-autoscaling:DescribeScalingActivities`
  - `application-autoscaling:DescribeScalingPolicies`
  - `application-autoscaling:DescribeScheduledActions`
  - `application-autoscaling:PutScalingPolicy`
  - `application-autoscaling:PutScheduledAction`
  - `application-autoscaling:RegisterScalableTarget`
- Pour mettre à jour les paramètres Time to Live avec l'`UpdateTimeToLive`API, vous devez disposer de l'autorisation suivante sur la ressource de table dans toutes les régions contenant des répliques :
  - `dynamodb:UpdateTimeToLive`

  Notez que le Time to Live (TTL) n'est pris en charge que pour les tables globales configurées avec MREC (Multi-Region Evental Consistency). Pour plus d'informations sur le fonctionnement des tables globales avec le protocole TTL, voir Fonctionnement des tables globales DynamoDB.

Supprimer des tables globales et supprimer des répliques

Pour supprimer une table globale, vous devez supprimer toutes les répliques. Les autorisations requises pour cette opération varient selon que vous supprimez une table globale avec ou sans région témoin.

Autorisations pour supprimer des tables globales et supprimer des répliques

Les autorisations suivantes sont requises à la fois pour supprimer des répliques individuelles et pour supprimer complètement des tables globales. La suppression d'une configuration de table globale supprime uniquement la relation de réplication entre les tables de différentes régions. Cela ne supprime pas la table DynamoDB sous-jacente dans la dernière région restante. La table de la dernière région continue d'exister en tant que table DynamoDB standard avec les mêmes données et paramètres. Ces autorisations s'appliquent à la fois aux tables globales de cohérence éventuelle multirégionale (MREC) et de cohérence forte entre régions (MRSC).

- Pour supprimer des répliques d'une table globale à l'aide de l'`UpdateTable`API, vous devez disposer de l'autorisation suivante sur la ressource de table de la région à partir de laquelle vous effectuez l'appel d'API :
  - `dynamodb:UpdateTable`

- Vous devez disposer des autorisations suivantes sur la ressource de table dans chaque région dans laquelle vous supprimez une réplique :
  - `dynamodb:DeleteTable`
  - `dynamodb:DeleteTableReplica`

Autorisations supplémentaires pour les tables globales MRSC utilisant un témoin

Pour supprimer une table globale multirégionale à forte cohérence (MRSC) avec un témoin, vous devez disposer de l'autorisation suivante sur la ressource de table dans toutes les régions participantes (y compris les régions répliques et la région témoin) :

- `dynamodb:DeleteGlobalTableWitness`

Exemples de politiques IAM pour supprimer les répliques d'une table globale

Suppression de répliques de tables globales

Cette politique basée sur l'identité vous permet de supprimer une table globale DynamoDB nommée « utilisateurs » et ses répliques dans trois régions :

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    }
  ]
```

```
}
```

Supprimer une table globale MRSC avec un témoin

Cette politique basée sur l'identité vous permet de supprimer la réplique et le témoin d'une table globale MRSC nommée « users » :

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica",
        "dynamodb:DeleteGlobalTableWitness"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    }
  ]
}
```

Utilisation des tables globales AWS KMS

Comme toutes les tables DynamoDB, les répliques de tables globales chiffrent toujours les données au repos à l'aide de clés de chiffrement stockées AWS dans Key Management Service ().AWS KMS

Tous les réplicas d'une table globale doivent être configurés avec le même type de clé KMS (clé détenue par AWS, clé gérée par AWS ou clé gérée par le client).

> **Important**
>
> DynamoDB a besoin d'accéder à la clé de chiffrement du réplica pour supprimer un réplica. Si vous souhaitez désactiver ou supprimer une clé gérée par le client utilisée pour chiffrer

un réplica parce que vous supprimez ce réplica, vous devez d'abord supprimer le réplica, attendre que le statut de la table de l'une des répliques restantes soit changé en `ACTIVE`, puis désactiver ou supprimer la clé.

Dans le cas d'une table globale configurée pour une cohérence finale multirégionale (MREC), si vous désactivez ou révoquez l'accès de DynamoDB à une clé gérée par le client utilisée pour chiffrer un réplica, la réplication vers et depuis le réplica s'arrêtera et le statut du réplica passera à `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Si un réplica d'une table globale MREC reste dans l'état `INACCESSIBLE_ENCRYPTION_CREDENTIALS` pendant plus de 20 heures, le réplica est converti de manière irréversible en une table DynamoDB à région unique.

Dans le cas d'une table globale configurée pour une forte cohérence multirégionale (MREC), si vous désactivez ou révoquez l'accès de DynamoDB à une clé gérée par le client utilisée pour chiffrer un réplica, la réplication vers et depuis le réplica s'arrêtera, les tentatives d'exécution d'écriture ou de lectures fortement cohérentes renverront une erreur, et l'état du réplica passera à `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Si un réplica d'une table globale MRSC reste dans l'état `INACCESSIBLE_ENCRYPTION_CREDENTIALS` pendant plus de sept jours, en fonction des autorisations spécifiques révoquées, le réplica sera archivé ou deviendra définitivement inaccessible.

Présentation de la facturation Amazon DynamoDB pour les tables globales

Ce guide décrit le fonctionnement de la facturation DynamoDB pour les tables globales, en identifiant les composants qui contribuent au coût des tables globales, y compris un exemple pratique.

Les tables globales Amazon DynamoDB sont une base de données entièrement gérée, sans serveur, multirégionale et multiactive. Les tables globales sont conçues pour une disponibilité de 99,999 %, offrant ainsi une résilience accrue des applications et une meilleure continuité des activités. Les tables globales répliquent automatiquement vos tables DynamoDB dans les régions de votre choix afin que vous puissiez obtenir AWS des performances de lecture et d'écriture rapides et locales.

Comment ça marche

Le modèle de facturation des tables globales est différent de celui des tables DynamoDB à région unique. Les opérations d'écriture pour les tables DynamoDB à région unique sont facturées selon les unités suivantes :

- Unités de demande d'écriture (WRUs) pour le mode capacité à la demande, où une WRU est facturée pour chaque écriture jusqu'à 1 Ko

- Unités de capacité d'écriture (WCUs) pour le mode de capacité provisionnée, où une WCU fournit une écriture par seconde pour un maximum de 1 Ko

Lorsque vous créez une table globale en ajoutant une table de réplicas à une table à région unique existante, cette table à région unique devient une table de réplicas, ce qui signifie que les unités utilisées pour facturer les écritures dans la table changent également. Les opérations d'écriture pour les tables de réplicas sont facturées selon les unités suivantes :

- Unités de demande d'écriture répliquées (rWRUs) pour le mode capacité à la demande, où une RWru par table de réplication est facturée pour chaque écriture jusqu'à 1 Ko
- Unités de capacité d'écriture répliquées (rWCUs) pour le mode de capacité provisionnée, où une WCU par table de réplication fournit une écriture par seconde pour un maximum de 1 Ko

Les mises à jour des index secondaires globaux (GSIs) sont facturées en utilisant les mêmes unités que les tables DynamoDB à région unique, même si la table de base du GSI est une table de réplique. Les opérations de mise à jour pour GSIs sont facturées selon les unités suivantes :

- Unités de demande d'écriture (WRUs) pour le mode capacité à la demande, où une WRU est facturée pour chaque écriture jusqu'à 1 Ko
- Unités de capacité d'écriture (WCUs) pour le mode de capacité provisionnée, où une WCU fournit une écriture par seconde pour un maximum de 1 Ko

Le prix des unités d'écriture répliquées (r WCUs et rWRUs) est le même que celui des unités d'écriture à région unique (WCUs et WRUs). Des frais de transfert de données entre régions s'appliquent aux tables globales, car les données sont répliquées dans toutes les régions. Des frais d'écriture répliquée (rWCU ou rWRU) sont facturés dans chaque région contenant une table de réplicas pour la table globale.

Les opérations de lecture à partir de tables à région unique et de tables de réplicas utilisent les unités suivantes :

- Unités de demande de lecture (RRUs) pour le mode de capacité à la demande, où une RRU est facturée pour chaque lecture hautement cohérente jusqu'à 4 Ko
- Unités de capacité de lecture (RCUs) pour les tables provisionnées, où une RCU fournit une lecture très cohérente par seconde jusqu'à 4 Ko

Modes de cohérence et facturation

Les unités d'écriture répliquées (r WCUs et rWRUs) utilisées pour facturer les opérations d'écriture sont identiques pour les modes de cohérence forte multirégion (MRSC) et de cohérence finale multirégionale (MREC). Les tables globales utilisant le mode MRSC (Multi-region Strong Cohérence) configuré avec un témoin n'entraînent pas de coûts unitaires d'écriture répliqués (r WCUs et rWRUs), de coûts de stockage ou de transfert de données pour la réplication vers le témoin.

Exemple de facturation des tables globales DynamoDB

Passons en revue un exemple de scénario de plusieurs jours pour voir comment fonctionne la facturation globale des demandes d'écriture de table dans la pratique (notez que cet exemple ne prend en compte que les demandes d'écriture et n'inclut pas les frais de restauration de tables et de transfert de données entre régions qui seraient encourus dans cet exemple) :

Jour 1 - Table à région unique : vous disposez d'une table DynamoDB à la demande à région unique nommée Table_A dans la région us-west-2. Vous écrivez 100 éléments de 1 Ko dans Table_A. Pour ces opérations d'écriture dans une seule région, 1 unité de demande d'écriture (WRU) vous est facturée par 1 Ko écrit. Vos frais pour le premier jour sont les suivants :

- 100 WRUs dans la région us-west-2 pour les écritures à région unique

Nombre total d'unités demandées facturées le jour 1 : 100 WRUs.

Jour 2 - Création d'une table globale : vous créez une table globale en ajoutant une réplica à Table_A dans la région us-east-2. Table_A est désormais une table globale avec deux tables de réplicas : un dans la région us-west-2 et l'autre dans la région us-east-2. Vous écrivez 150 éléments de 1 Ko dans la table de réplicas de la région us-west-2. Vos frais pour le deuxième jour sont les suivants :

- 150 r WRUs dans la région us-west-2 pour les écritures répliquées
- 150 r WRUs dans la région us-east-2 pour les écritures répliquées

Nombre total d'unités demandées facturées le jour 2 : 300 WRUs r.

Jour 3 - Ajout d'un index secondaire global : Vous ajoutez un index secondaire global (GSI) à la table de réplicas de la région us-east-2 qui projette tous les attributs de la table de base (réplica). La table globale crée automatiquement pour vous le GSI sur la table de réplicas de la région us-west-2. Vous écrivez 200 nouveaux enregistrements de 1 Ko dans la table de réplicas de la région us-west-2. Vos frais pour le troisième jour sont les suivants :

- • 200 r WRUs dans la région us-west-2 pour les écritures répliquées
- • 200 WRUs dans la région us-west-2 pour les mises à jour du GSI
- • 200 r WRUs dans la région us-east-2 pour les écritures répliquées
- • 200 WRUs dans la région us-east-2 pour les mises à jour du GSI

Nombre total d'unités de demande d'écriture facturées le jour 3 : 400 WRUs et 400 WRUs r.

Les frais unitaires d'écriture totaux pour les trois jours sont de 500 WRUs (100 WRU le jour 1 + 400 WRUs le jour 3) et de 700 r WRUs (300 r WRUs le jour 2 + 400 r WRUs le jour 3).

En résumé, les opérations d’écriture de tables de réplicas sont facturées en unités d’écriture répliquées dans toutes les régions qui contiennent une table de réplicas. Si vous avez des index secondaires globaux, des unités d'écriture vous sont facturées pour les mises à jour GSIs dans toutes les régions contenant un GSI (dans une table globale, il s'agit de toutes les régions contenant une table répliquée).

Versions des tables globales DynamoDB

Deux versions des tables globales DynamoDB sont disponibles : la version 2019.11.21 (actuelle) et la version 2017.11.29 (héritée). Nous recommandons d’utiliser la version 2019.11.21 (actuelle) des tables globales, car elle est plus facile à utiliser, est prise en charge dans un plus grand nombre de régions et coûte moins cher pour la plupart des charges de travail que la version 2017.11.29 (héritée).

Détermination de la version d’une table globale

Déterminer la version à l'aide du AWS CLI

Identification d’une réplica de table globale de la version 2019.11.21 (actuelle)

Pour déterminer si une table est une réplica de table globale 2019.11.21 (actuelle), appelez la commande `describe-table` de la table. Si la sortie contient l’attribut `GlobalTableVersion` dont la valeur est « 2019.11.21 », la table est un réplica de table globale de la version 2019.11.21 (actuelle).

Exemple de commande CLI pour `describe-table` :

```
aws dynamodb describe-table \
--table-name users \
```

```
--region us-east-2
```

La sortie (abrégée) contient l'attribut `GlobalTableVersion` dont la valeur est « 2019.11.21 ». La table est donc un réplica de table globale de la version 2019.11.21 (actuelle).

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

Identification d'un réplica de table version 2017.11.29 (héritée)

La version 2017.11.29 (héritée) des tables globales utilise un ensemble de commandes dédié pour la gestion globale des tables. Pour déterminer si une table est un réplica de tables globales version 2017.11.29 (actuelle), invoquez la commande `describe-global-table` pour la table. Si vous recevez une réponse positive, la table est un réplica de table globale de la version 2017.11.29 (héritée). Si la commande `describe-global-table` renvoie une erreur `GlobalTableNotFoundException`, la table n'est pas un réplica de la version 2017.11.29 (héritée).

Exemple de commande CLI pour `describe-global-table` :

```
aws dynamodb describe-global-table \
```

```
--table-name users \
--region us-east-2
```

La commande renvoie une réponse positive, de sorte que la table est un réplica de table globale de la version 2017.11.29 (héritée).

```
{
    "GlobalTableDescription": {
        "ReplicationGroup": [
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
    }
}
```

Détermination de la version à l'aide de la console DynamoDB

Pour identifier la version d'un réplica de table globale, effectuez les opérations suivantes :

1. Ouvrez la console DynamoDB à la maison. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez la table dont vous souhaitez identifier la version des tables globales.
4. Choisissez l'onglet Tables globales.

   La section Récapitulatif affiche la version des tables globales utilisée.

Différences de comportement entre les versions héritées et les versions actuelles

La liste suivante décrit les différences de comportement entre les versions héritées et actuelles des tables globales.

- La version 2019.11.21 (actuelle) utilise moins de capacité d'écriture pour plusieurs opérations DynamoDB que la version 2017.11.29 (héritée), et est donc plus rentable pour la plupart des clients. Les différences de ces opérations DynamoDB sont les suivantes :
  - L'appel PutItemd'un élément de 1 Ko dans une région et sa réplication vers d'autres régions nécessitent 2 r WRUs par région pour le 29 novembre 2017 (ancien), mais seulement 1 RWru pour le 21 novembre 2019 (actuel).
  - L'invocation UpdateItemd'un élément de 1 Ko nécessite 2 r WRUs dans la région source et 1 RWru par région de destination pour le 29 novembre 2017 (ancienne), mais un seul RWru pour les régions source et de destination pour le 21 novembre 2019 (actuel).
  - L'appel DeleteItemd'un élément de 1 Ko nécessite 1 RWru dans la région source et 2 r WRUs par région de destination pour le 29 novembre 2017 (ancien), mais un seul RWru pour la région source ou de destination pour le 21 novembre 2019 (actuel).

  Le tableau suivant montre la consommation rWRU des tables 2017.11.29 (ancienne) et 2019.11.21 (actuelle) pour un élément de 1 Ko dans deux régions.

Opération	2017.11.29 (héritée)	2019.11.21 (actuelle)	Économies
PutItem	4 r WRUs	2 r WRUs	50%
UpdateItem	3 r WRUs	2 r WRUs	33 %
DeleteItem	3 r WRUs	2 r WRUs	33 %

- la version 2017.11.29 (Legacy) n'est disponible que dans la version 11. Régions AWS Cependant, la version 2019.11.21 (actuelle) est disponible dans toutes les Régions AWS.
- Vous créez les tables globales de la version 2017.11.29 (Legacy) en créant d'abord un ensemble de tables régionales vides, puis en invoquant l'CreateGlobalTableAPI pour former la table globale. Vous créez des tables globales de la version 2019.11.21 (Current) en appelant l'UpdateTableAPI pour ajouter une réplique à une table régionale existante.
- La version 2017.11.29 (héritée) vous oblige à vider tous les réplicas de la table avant d'ajouter un réplica dans une nouvelle région (y compris lors de la création). La version 2019.11.21 (actuelle) vous permet d'ajouter et de supprimer des réplicas dans les régions d'une table contenant déjà des données.
- la version 2017.11.29 (Legacy) utilise l'ensemble de plans de contrôle dédié suivant APIs pour gérer les répliques :

- CreateGlobalTable
- DescribeGlobalTable
- DescribeGlobalTableSettings
- ListGlobalTables
- UpdateGlobalTable
- UpdateGlobalTableSettings

la version 2019.11.21 (Current) utilise le DescribeTableet UpdateTable APIs pour gérer les répliques.

- la version 2017.11.29 (héritée) publie deux enregistrements DynamoDB Streams pour chaque écriture. La version 2019.11.21 (actuelle) ne publie qu'un seul enregistrement DynamoDB Streams pour chaque écriture.
- La version 2017.11.29 (héritée) renseigne et met à jour les attributs `aws:rep:deleting`, `aws:rep:updateregion` et `aws:rep:updatetime`. La version 2019.11.21 (actuelle) ne renseigne ni ne met à jour ces attributs.
- La version 2017.11.29 (héritée) ne synchronise pas les paramètres Utilisation de la durée de vie (TTL) dans DynamoDB entre les réplicas. La version 2019.11.21 (actuelle) synchronise les paramètres TTL entre les réplicas.
- la version 2017.11.29 (héritée) ne réplique pas les suppressions TTL vers d'autres réplicas. La version 2019.11.21 (actuelle) réplique les suppressions TTL dans tous les réplicas.
- La version 2017.11.29 (héritée) ne synchronise pas les paramètres d'autoscaling entre les réplicas. La version 2019.11.21 (actuelle) synchronise les paramètres d'autoscaling entre les réplicas.
- La version 2017.11.29 (héritée) ne synchronise pas les paramètres d'index secondaires globaux (GSI) entre les réplicas. La version 2019.11.21 (actuelle) synchronise les paramètres GSI entre les réplicas.
- La version 2017.11.29 (héritée) ne synchronise pas les paramètres de chiffrement au repos entre les réplicas. La version 2019.11.21 (actuelle) synchronise les paramètres de chiffrement au repos entre les réplicas.
- La version 2017.11.29 (héritée) publie la métrique `PendingReplicationCount`. La version 2019.11.21 (actuelle) ne publie pas cette métrique.

Mise à niveau vers la version actuelle

Autorisations requises pour la mise à niveau des tables globales

Pour effectuer une mise à niveau vers la version 2019.11.21 (actuelle), vous devez disposer d'autorisations `dynamodb:UpdateGlobalTableversion` dans toutes les régions avec des réplicas. Ces autorisations s'ajoutent aux autorisations nécessaires pour accéder à la console DynamoDB et afficher les tables.

La politique IAM suivante accorde des autorisations pour mettre à niveau n'importe quelle table globale vers la version 2019.11.21 (actuelle).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

La politique IAM suivante accorde des autorisations pour mettre à niveau uniquement la table globale `Music`, avec des réplicas dans deux régions, vers la version 2019.11.21 (actuelle).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
            ]
        }
    ]
}
```

À quoi s’attendre lors de la mise à niveau

- Tous les réplicas de tables globales continueront à traiter le trafic de lecture et d’écriture pendant la mise à niveau.
- Le processus de mise à niveau prend entre quelques minutes et plusieurs heures selon la taille de la table et le nombre de réplicas.
- Au cours du processus de mise à niveau, la valeur de TableStatuspassera de `ACTIVE` à`UPDATING`. Vous pouvez consulter l'état de la table en appelant l'DescribeTableAPI ou en utilisant la vue Tables de la console DynamoDB.
- L’autoscaling n’ajustera pas les paramètres de capacité provisionnée pour une table globale pendant la mise à niveau de la table. Nous vous recommandons vivement de configurer la table en mode capacité à la demande lors de la mise à niveau.
- Si vous choisissez d’utiliser la capacité provisionné avec autoscaling pendant la mise à niveau, vous devez augmenter le débit minimum de lecture et d’écriture de vos politiques pour tenir compte des augmentations attendues du trafic et éviter toute limitation pendant la mise à niveau.
- La métrique `ReplicationLatency` peut signaler temporairement les pics de latence ou arrêter de signaler les données métriques pendant le processus de mise à niveau. Pour plus d’informations, consultez the section called “ReplicationLatency”.
- Lorsque le processus de mise à niveau est terminé, l’état de votre table revient à `ACTIVE`.

Comportement de DynamoDB Streams avant, pendant et après la mise à niveau

Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
Placement ou mise à jour	Source	La populatio n d'horodat age se produit en utilisant. UpdateItem	La population d'horodatage se produit en utilisant. PutItem	Aucun horodatag e visible par le client n’est généré.
		Deux enregistr ements de flux sont générés. Le premier	Deux enregistr ements de flux sont générés. Le premier	Un seul enregistr ement de flux contenant les attributs écrits

Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
		enregistrement contient les attributs écrits par le client. Le deuxième enregistrement contient les attributs `aws:rep:*` .	enregistrement contient les attributs écrits par le client. Le deuxième enregistrement contient les attributs `aws:rep:*` .	par le client est généré.
		Deux r WCUs sont consommés pour chaque écriture du client.	Deux r WCUs sont consommés pour chaque écriture du client.	Une rWCU est utilisée pour chaque écriture du client.
		`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` la métrique est publiée dans CloudWatch.
	Destination	La réplication s'effectue à l'aide de PutItem.	La réplication s'effectue à l'aide de PutItem.	La réplication s'effectue à l'aide de PutItem.

Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
		Un seul enregistrement de flux est généré, qui contient à la fois les attributs écrits par le client et les attributs `aws:rep:*` .	Un seul enregistrement de flux est généré, qui contient à la fois les attributs écrits par le client et les attributs `aws:rep:*` .	Un seul enregistrement de flux est généré, qui contient uniquement les attributs écrits par le client et aucun attributs de réplication.
		Une rWCU est utilisée si l'élément existe dans la région de destination. Deux r WCUs sont consommés si l'article n'existe pas dans la région de destination.	Une rWCU est utilisée si l'élément existe dans la région de destination. Deux r WCUs sont consommés si l'article n'existe pas dans la région de destination.	Une rWCU est utilisée pour chaque écriture du client.
		`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` la métrique est publiée dans CloudWatch.

Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
Suppression	Source	Supprimez tout élément dont l'horodatage est plus petit en utilisant. DeleteItem	Supprimez tout élément dont l'horodatage est plus petit en utilisant. DeleteItem	Supprimez tout élément dont l'horodatage est plus petit en utilisant. DeleteItem
		Un seul enregistrement de flux est généré, qui contient à la fois les attributs écrits par le client et les attributs `aws:rep:*` .	Un seul enregistrement de flux est généré, qui contient à la fois les attributs écrits par le client et les attributs `aws:rep:*` .	Un seul enregistrement de flux est généré, qui contient les attributs écrits par le client.
		Une rWCU est utilisée pour chaque suppression du client.	Une rWCU est utilisée pour chaque suppression du client.	Une rWCU est utilisée pour chaque suppression du client.
		`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` la métrique est publiée dans CloudWatch.

Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
	Destination	Des suppressions en deux phases ont lieu : • Dans la phase 1, UpdateItem définit l'indicateur de suppression. • Dans la phase 2, DeleteItem supprime l'élément.	Supprime l'élément à l'aide DeleteItem de.	Supprime l'élément à l'aide DeleteItem de.
		Deux enregistrements de flux sont générés. Le premier enregistrement contient la modification apportée au champ `aws:rep:deleting`. Le deuxième enregistrement contient les attributs écrits par le client et les attributs `aws:rep:*`.	Un seul enregistrement de flux est généré, qui contient les attributs écrits par le client.	Un seul enregistrement de flux est généré, qui contient les attributs écrits par le client.

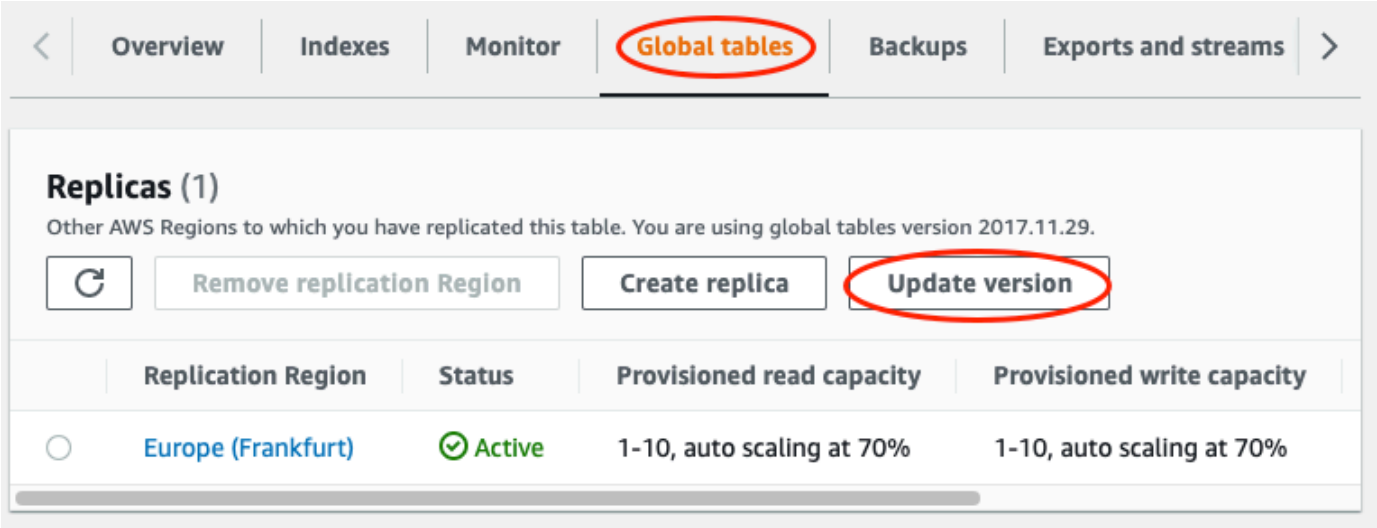
Opération	Région de réplica	Comportement avant la mise à niveau	Comportement pendant la mise à niveau	Comportement après la mise à niveau
		Deux r WCUs sont consommés pour chaque suppression par un client.	Une rWCU est utilisée pour chaque suppression du client.	Une rWCU est utilisée pour chaque suppression du client.
		`ReplicationLatency` et `PendingReplicationCount` les statistiques sont publiées dans CloudWatch.	`ReplicationLatency` la métrique est publiée dans CloudWatch.	`ReplicationLatency` la métrique est publiée dans CloudWatch.

Mise à jour vers la version 2019.11.21 (actuelle)

Procédez comme suit pour mettre à jour la version des tables globales DynamoDB à l'aide de la AWS Management Console.

Pour mettre à niveau des tables globales vers la version 2019.11.21 (actuelle)

1. Ouvrez la console DynamoDB à la maison. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation situé sur le côté gauche de la console, choisissez Tables, puis sélectionnez la table globale à mettre à niveau vers la version 2019.11.21 (actuelle).
3. Choisissez l'onglet Tables globales.
4. Choisissez Update version (Mettre à jour la version).



5. Lisez et acceptez les nouvelles exigences, puis choisissez Mettre à jour la version.
6. Une fois le processus de mise à niveau terminé, la version des tables globales qui apparaît sur la console est la version 2019.11.21.

Bonnes pratiques relatives aux tables globales

Les sections suivantes décrivent les bonnes pratiques en matière de déploiement et d'utilisation de tables globales.

Version

Deux versions des tables globales DynamoDB sont disponibles : la version 2019.11.21 (actuelle) et la version 2017.11.29 (héritée). Vous devez utiliser la version 2019.11.21 (actuelle) dans la mesure du possible.

Protection contre la suppression

Vous devez activer la protection contre la suppression sur les réplicas de tables globales que vous souhaitez protéger contre toute suppression accidentelle. Vous devez activer la protection contre la suppression sur chaque réplica.

Utilisation de AWS CloudFormation

CloudFormation ne prend actuellement pas en charge la coordination des ressources multirégionales telles que les tables globales entre les piles. Si vous définissez chaque réplica d'une table globale dans une pile régionale distincte, vous rencontrerez des erreurs dues à la détection d'une dérive

entre les piles lors de la mise à jour des réplicas. Pour éviter ce problème, vous devez choisir une région comme région de référence pour déployer vos tables globales et définir tous les réplicas de vos tables globales dans la pile de cette région.

> **Important**
>
> Vous ne pouvez pas convertir une ressource de type `AWS::DynamoDB::Table` en une ressource de type `AWS::DynamoDB::GlobalTable` en modifiant son type dans votre modèle. Toute tentative de conversion d'une table à région unique en table globale en modifiant son type de ressource CloudFormation peut entraîner la suppression de votre table DynamoDB.

Vous pouvez utiliser la ressource `AWS::DynamoDB::GlobalTable` pour créer une table dans une seule région. Cette table sera déployée comme toute autre table à région unique. Si vous mettez ultérieurement à jour la pile pour ajouter d'autres régions à une ressource, des réplicas seront ajoutés à la table et celle-ci sera convertie en toute sécurité en table globale.

Si vous souhaitez convertir une ressource `AWS::DynamoDB::Table` existante en ressource `AWS::DynamoDB::GlobalTable`, les étapes recommandées pour convertir le type de ressource sont les suivantes :

1. Définir la politique de suppression `AWS::DynamoDB::Table` à conserver.
2. Retirer la table de la définition de pile.
3. Ajouter des réplicas à la table à région unique de la console AWS, en la convertissant en table globale.
4. Importer la table globale en tant que nouvelle ressource `AWS::DynamoDB::GlobalTable` dans la pile.

Sauvegardes et reprise ponctuelle

L'activation des sauvegardes automatisées et de la reprise ponctuelle (PITR) pour un réplica dans une table globale peut être suffisante pour atteindre vos objectifs de reprise après sinistre. Les sauvegardes de réplicas créées avec AWS-Backup peuvent être automatiquement répliquées entre les régions pour une meilleure résilience. Tenez compte des objectifs de votre plan de reprise après sinistre dans le contexte de la haute disponibilité multirégionale lorsque vous choisissez votre stratégie de sauvegarde et d'activation du PITR.

Conception pour une haute disponibilité multirégionale

Pour obtenir des conseils prescriptifs sur le déploiement de tables globales, consultez les Bonnes pratiques relatives à la conception d'une table globale DynamoDB.

Utilisation d'éléments et d'attributs dans DynamoDB

Dans Amazon DynamoDB, un élément est une collection d'attributs. Chaque attribut a un nom et une valeur. Une valeur d'attribut peut être de type scalaire, d'ensemble ou de document. Pour de plus amples informations, veuillez consulter Fonctionnement d'Amazon DynamoDB.

DynamoDB fournit quatre opérations correspondant aux fonctionnalités de base de création, lecture, mise à jour et suppression (CRUD). Toutes ces opérations sont atomiques.

- `PutItem` – Création d'un élément
- `GetItem` – Lecture d'un élément
- `UpdateItem` – Mise à jour d'un élément
- `DeleteItem` – Suppression d'un élément

Pour chacune de ces opérations, vous devez indiquer la clé primaire de l'élément que vous souhaitez utiliser. Par exemple, pour lire un élément avec `GetItem`, vous devez spécifier la clé de partition et la clé de tri (le cas échéant) de cet élément.

Outre les quatre opérations CRUD de base, DynamoDB offre les fonctionnalités suivantes :

- `BatchGetItem` – Lecture de 100 éléments au plus à partir d'une ou de plusieurs tables
- `BatchWriteItem` – Création ou suppression de 25 éléments au plus dans une ou plusieurs tables

Ces opérations par lots peuvent associer plusieurs opérations CRUD dans une même demande. En outre, les opérations par lots lisent et écrivent des éléments en parallèle afin de réduire les latences de réponse.

Cette section explique comment utiliser ces opérations. Elle comprend des rubriques connexes telles que les mises à jour conditionnelles et les compteurs atomiques. Cette section inclut également un exemple de code utilisant le AWS SDKs.

Rubriques

- Tailles et formats d'élément DynamoDB
- Lecture d'un élément
- Écriture d'un élément
- Valeurs renvoyées
- Opérations par lots
- Compteurs atomiques
- Écritures conditionnelles
- Utilisation d'expressions dans DynamoDB
- Utilisation de la durée de vie (TTL) dans DynamoDB
- Interrogation de tables dans DynamoDB
- Analyse de tables dans DynamoDB
- PartiQL – Langage de requête compatible SQL pour Amazon DynamoDB
- Utilisation d'éléments : Java
- Utilisation d'élements : .NET

Tailles et formats d'élément DynamoDB

Les tables DynamoDB étant sans schéma, sauf pour la clé primaire, les éléments d'une table peuvent avoir des attributs, tailles et types de données différents.

La taille totale d'un élément est la somme des longueurs de ses noms et valeurs d'attribut, plus toute surcharge applicable telle que décrite ci-dessous. Vous pouvez évaluer la taille des attributs à l'aide des consignes suivantes :

- Les chaînes sont Unicode avec codage binaire UTF-8. La taille d'une chaîne est (nombre d'octets codés en UTF-8 du nom d'attribut) + (nombre d'octets codés en UTF-8).
- Les nombres sont de longueur variable, avec 38 chiffres significatifs au plus. Les zéros de début et de fin sont tronqués. La taille d'un nombre est approximativement (nombre d'octets codés en UTF-8 du nom d'attribut) + (1 octet pour 2 chiffres significatifs) + (1 octet).
- Une valeur binaire doit être codée en base 64 avant d'être envoyée à DynamoDB, mais la longueur en octets bruts de la valeur est utilisée pour le calcul de la taille. La taille d'un binary attribute est (nombre d'octets codés en UTF-8 du nom d'attribut) + (nombre d'octets bruts).
- La taille d'un attribut nul ou booléen est (nombre d'octets codés en UTF-8 du nom d'attribut) + (1 octet).

- Un attribut de type `List` ou `Map` a besoin de 3 octets de surcharge, quel que soit son contenu. La taille d'un attribut `List` ou `Map` est (nombre d'octets codés en UTF-8 du nom d'attribut) + somme de (taille des éléments imbriqués) + (3 octets). La taille d'un attribut `List` ou `Map` vide est (nombre d'octets codés en UTF-8 du nom d'attribut) + (3 octets).
- Chaque élément `List` ou `Map` nécessite également 1 octet de surcharge.

> **Note**
>
> Nous vous recommandons de choisir des noms d'attribut courts. Cela vous permet de réduire la quantité de stockage requise, mais peut également réduire la quantité RCU/WCUs que vous utilisez.

À des fins de facturation du stockage, chaque élément inclut une surcharge de stockage par élément qui dépend des fonctionnalités que vous avez activées.

- Tous les éléments de DynamoDB nécessitent une surcharge de stockage de 100 octets pour l'indexation.
- Certaines fonctionnalités DynamoDB (tables globales, transactions, récupération de données de modification pour Kinesis Data Streams avec DynamoDB) nécessitent une surcharge de stockage supplémentaire pour tenir compte des attributs créés par le système résultant de l'activation de ces fonctions. Par exemple, les tables globales nécessitent une surcharge de stockage de 48 octets supplémentaires.

Lecture d'un élément

L'opération `GetItem` vous permet de lire un élément à partir d'une table DynamoDB. Vous devez fournir le nom de la table, ainsi que la clé primaire de l'élément souhaité.

Example

L'AWS CLI exemple suivant montre comment lire un élément du `ProductCatalog` tableau.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Avec `GetItem`, vous devez spécifier la totalité de la clé primaire, et non uniquement une partie. Par exemple, si une table possède une clé primaire composite (clé de partition et clé de tri), vous devez fournir une valeur pour la clé de partition et une valeur pour la clé de tri.

Une demande `GetItem` effectue une lecture cohérente à terme par défaut. Le paramètre `ConsistentRead` vous permet de demander une lecture fortement cohérente à la place. (Cela consomme des unités de capacité de lecture supplémentaires, mais renvoie la up-to-date version la plus complète de l'article.)

`GetItem` renvoie tous les attributs de l'élément. Vous pouvez renvoyer uniquement certains des attributs à l'aide d'une expression de projection. Pour de plus amples informations, veuillez consulter Utilisation d'expressions de projection dans DynamoDB.

Pour renvoyer le nombre d'unités de capacité de lecture consommées par l'opération `GetItem`, définissez le paramètre `ReturnConsumedCapacity` sur la valeur `TOTAL`.

Example

L'exemple suivant AWS Command Line Interface(AWS CLI) montre certains des `GetItem` paramètres facultatifs.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Écriture d'un élément

Créez, mettez à jour ou supprimez un élément dans une table DynamoDB à l'aide de l'une des opérations suivantes :

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Pour chacune de ces opérations, vous devez spécifier la totalité de la clé primaire, et non uniquement une partie. Par exemple, si une table possède une clé primaire composite (clé de partition et clé de tri), vous devez fournir une valeur pour la clé de partition et une valeur pour la clé de tri.

Pour renvoyer le nombre d'unités de capacité d'écriture consommées par l'une de ces opérations, définissez le paramètre `ReturnConsumedCapacity` sur l'une des valeurs suivantes :

- `TOTAL` – Renvoie le nombre total d'unités de capacité d'écriture consommées.
- `INDEXES` – Renvoie le nombre total d'unités de capacité de lecture consommées, avec les sous-totaux correspondant à la table et aux index secondaires concernés par l'opération.
- `NONE` – Aucun détail concernant la capacité d'écriture n'est renvoyé. (Il s'agit de l'option par défaut.)

PutItem

`PutItem` crée un élément. Si un élément avec la même clé existe déjà dans la table, il est remplacé par le nouvel élément.

Example

Écrivez un nouvel élément dans la table `Thread`. La clé primaire de la table `Thread` est composée de `ForumName` (clé de partition) et `Subject` (clé de tri).

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

Les arguments de la fonction `--item` sont stockés dans le fichier `item.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

UpdateItem

S'il n'existe aucun élément associé à la clé spécifiée, `UpdateItem` en crée un. Dans le cas contraire, l'opération modifie les attributs d'un élément existant.

Vous préciser les attributs à modifier, ainsi que leurs nouvelles valeurs, à l'aide d'une expression de mise à jour. Pour de plus amples informations, veuillez consulter Utilisation d'expressions de mise à jour dans DynamoDB.

Au sein de l'expression de mise à jour, vous utilisez des valeurs d'attributs d'expression comme espaces réservés pour les valeurs réelles. Pour de plus amples informations, veuillez consulter Utilisation de valeurs d'attributs d'expression dans DynamoDB.

Example

Modifiez plusieurs attributs dans l'élément `Thread`. Le paramètre `ReturnValues` facultatif présente l'élément tel qu'il s'affiche après la mise à jour. Pour de plus amples informations, veuillez consulter Valeurs renvoyées.

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

Les arguments de la fonction `--key` sont stockés dans le fichier `key.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `expression-attribute-values.json`.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

`DeleteItem` supprime l'élément avec la clé spécifiée.

Example

L'AWS CLI exemple suivant montre comment supprimer l'`Thread`élément.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Valeurs renvoyées

Dans certains cas, vous souhaitez que DynamoDB renvoie certaines valeurs d'attribut telles qu'elles s'affichent avant ou après leur modification. Les opérations `PutItem`, `UpdateItem` et `DeleteItem` ont un paramètre `ReturnValues` vous permettant de renvoyer les valeurs d'attribut avant ou après leur modification.

La valeur par défaut de `ReturnValues` est `NONE`. Autrement dit, DynamoDB ne renvoie aucune information concernant les attributs modifiés.

Les autres paramètres valides pour `ReturnValues` sont les suivants, classés par opération d'API DynamoDB.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Si vous remplacez un élément existant, `ALL_OLD` renvoie l'élément entier tel qu'il s'affichait avant son remplacement.
  - Si vous écrivez un élément qui n'existe pas, `ALL_OLD` n'a aucun effet.

UpdateItem

L'utilisation la plus courante d'`UpdateItem` est la mise à jour d'un élément existant. Cependant, `UpdateItem` effectue une opération de mise à jour/insertion. Autrement dit, il crée automatiquement l'élément si celui-ci n'existe pas.

- `ReturnValues`: `ALL_OLD`

- Si vous mettez à jour un élément existant, `ALL_OLD` renvoie l'élément entier tel qu'il s'affichait avant sa mise à jour.
- Si vous mettez à jour un élément qui n'existe pas (upsert), `ALL_OLD` n'a aucun effet.

- `ReturnValues`: `ALL_NEW`
  - Si vous mettez à jour un élément existant, `ALL_NEW` renvoie l'élément entier tel qu'il s'affiche après sa mise à jour.
  - Si vous mettez à jour un élément qui n'existe pas (upsert), `ALL_NEW` renvoie l'élément entier.
- `ReturnValues`: `UPDATED_OLD`
  - Si vous mettez à jour un élément existant, `UPDATED_OLD` renvoie uniquement les attributs mis à jour tels qu'ils s'affichaient avant leur mise à jour.
  - Si vous mettez à jour un élément qui n'existe pas (upsert), `UPDATED_OLD` n'a aucun effet.
- `ReturnValues`: `UPDATED_NEW`
  - Si vous mettez à jour un élément existant, `UPDATED_NEW` renvoie uniquement les attributs affectés tels qu'ils s'affichaient après leur mise à jour.
  - Si vous mettez à jour (upsert) un élément qui n'existe pas, `UPDATED_NEW` renvoie uniquement les attributs mis à jour tels qu'ils s'affichent après la mise à jour.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Si vous supprimez un élément existant, `ALL_OLD` renvoie l'élément entier tel qu'il s'affichait avant sa suppression.
  - Si vous supprimez un élément qui n'existe pas, `ALL_OLD` ne renvoie pas de données.

Opérations par lots

Pour les applications ayant besoin de lire ou d'écrire plusieurs éléments, DynamoDB fournit les opérations `BatchGetItem` et `BatchWriteItem`. L'utilisation de ces opérations peut réduire le nombre d'allers et retours réseau entre votre application et DynamoDB. En outre, DynamoDB effectue les opérations de lecture ou d'écriture individuelles en parallèle. Vos applications bénéficient de ce parallélisme sans devoir gérer la simultanéité ou les threads.

Les opérations par lots sont essentiellement des wrappers autour de plusieurs demandes de lecture ou d'écriture. Par exemple, si une demande `BatchGetItem` comporte cinq éléments, DynamoDB

exécute cinq opérations `GetItem` pour vous. De même, si une demande `BatchWriteItem` comporte deux demandes d'insertion et quatre demandes de suppression, DynamoDB exécute deux demandes `PutItem` et quatre demandes `DeleteItem`.

En général, une opération par lots n'échoue pas, sauf si toutes les demandes du lot échouent. Supposons, par exemple, que vous effectuiez une opération `BatchGetItem`, mais que l'une des demandes `GetItem` du lot échoue. Dans ce cas, `BatchGetItem` renvoie les clés et les données provenant de la demande `GetItem` qui a échoué. Les autres demandes `GetItem` du lot ne sont pas affectées.

BatchGetItem

Une seule opération `BatchGetItem` peut contenir jusqu'à 100 demandes `GetItem` individuelles, et peuvent extraire jusqu'à 16 Mo de données. En outre, une opération `BatchGetItem` peut extraire des éléments de plusieurs tables.

Example

Extrayez deux éléments de la table `Thread` en renvoyant uniquement certains des attributs à l'aide d'une expression de projection.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Les arguments de la fonction `--request-items` sont stockés dans le fichier `request-items.json`.

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
```

```
}
```

BatchWriteItem

L'opération `BatchWriteItem` peut contenir jusqu'à 25 demandes `PutItem` et `DeleteItem` individuelles, et peut écrire jusqu'à 16 Mo de données (La taille maximale d'un élément est de 400 Ko.) En outre, une même opération `BatchWriteItem` peut insérer ou supprimer des éléments de plusieurs tables.

> **Note**
>
> `BatchWriteItem` ne prend pas en charge les demandes `UpdateItem`.

Example

Ajoutez deux éléments à la table `ProductCatalog`.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Les arguments de la fonction `--request-items` sont stockés dans le fichier `request-items.json`.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
```

```
                }
            }
        }
    ]
}
```

Compteurs atomiques

L'opération `UpdateItem` vous permet d'implémenter un compteur atomique, c'est-à-dire un attribut numérique incrémenté sans condition, sans interférer avec d'autres demandes d'écriture. (Toutes les demandes d'écriture sont appliquées dans l'ordre dans lequel elles ont été reçues.) Avec un compteur atomique, les mises à jour ne sont pas idempotentes. Autrement dit, la valeur numérique augmente ou diminue chaque fois que vous appelez `UpdateItem`. Si la valeur d'incrément permettant de mettre à jour le compteur atomique est positive, cela peut provoquer un surcomptage. Si la valeur d'incrément est négative, cela peut entraîner une sous-estimation.

Un compteur atomique vous permet de suivre le nombre de visiteurs d'un site web. Dans ce cas, l'application doit incrémenter une valeur numérique, quelle que soit sa valeur actuelle. En cas d'échec d'une opération `UpdateItem`, l'application peut simplement retenter l'opération. Cela risquerait de mettre à jour le compteur deux fois, mais vous pourriez probablement tolérer de sur-comptabiliser ou sous-comptabiliser légèrement les visiteurs.

Dans le cas contraire, l'utilisation d'un compteur automatique n'est pas adaptée (par exemple dans une application bancaire). Il est alors plus sûr d'utiliser une mise à jour conditionnelle plutôt qu'un compteur atomique.

Pour de plus amples informations, veuillez consulter Incrémentation et décrémentation d'attributs numériques.

Example

L'AWS CLI exemple suivant augmente la valeur `Price` d'un produit de 5. Dans cet exemple, l'existence de l'élément était connue avant la mise à jour du compteur. Puisque l'opération `UpdateItem` n'est pas idempotente, le `Price` augmente à chaque exécution de ce code.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
```

```
    --return-values UPDATED_NEW
```

Écritures conditionnelles

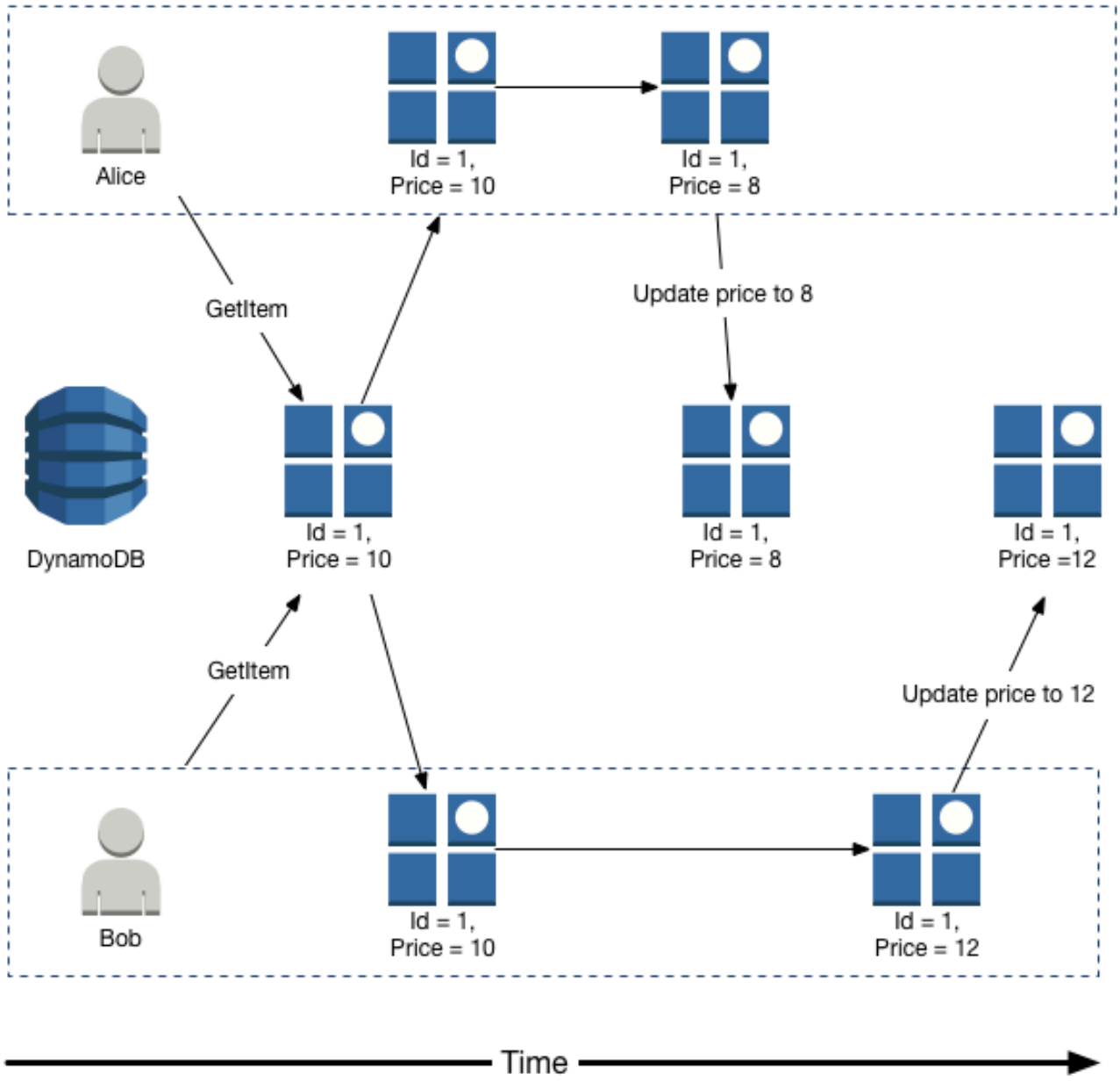
Par défaut, les opérations d'écriture de DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) sont sans condition : chacune remplace un élément existant doté de la clé primaire spécifiée.

DynamoDB prend en charge, en option, les écritures conditionnelles pour ces opérations. Une écriture conditionnelle réussit seulement si les attributs de l'élément remplissent une ou plusieurs conditions prévues. Dans le cas contraire, elle renvoie une erreur.

Les écritures conditionnelles vérifient leurs conditions par rapport à la dernière version mise à jour de l'élément. Notez que si l'élément n'existait pas auparavant ou si la dernière opération réussie sur cet élément était une suppression, l'écriture conditionnelle ne trouvera aucun élément précédent.

Les écritures conditionnelles sont utiles dans de nombreux cas. Par exemple, vous pouvez décider qu'une opération `PutItem` réussit seulement si aucun élément avec la même clé primaire n'existe déjà. Vous pouvez également empêcher une opération `UpdateItem` de modifier un élément si l'un de ses attributs possède une valeur donnée.

Les écritures conditionnelles sont utiles dans les cas où plusieurs utilisateurs tentent de modifier le même élément. Examinez le diagramme suivant, dans lequel deux utilisateurs (Alice et Bob) utilisent le même élément provenant d'une table DynamoDB.



Supposons qu'Alice utilise le AWS CLI pour mettre à jour l'`Price`attribut à 8.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `expression-attribute-values.json` :

```
{
    ":newval":{"N":"8"}
}
```

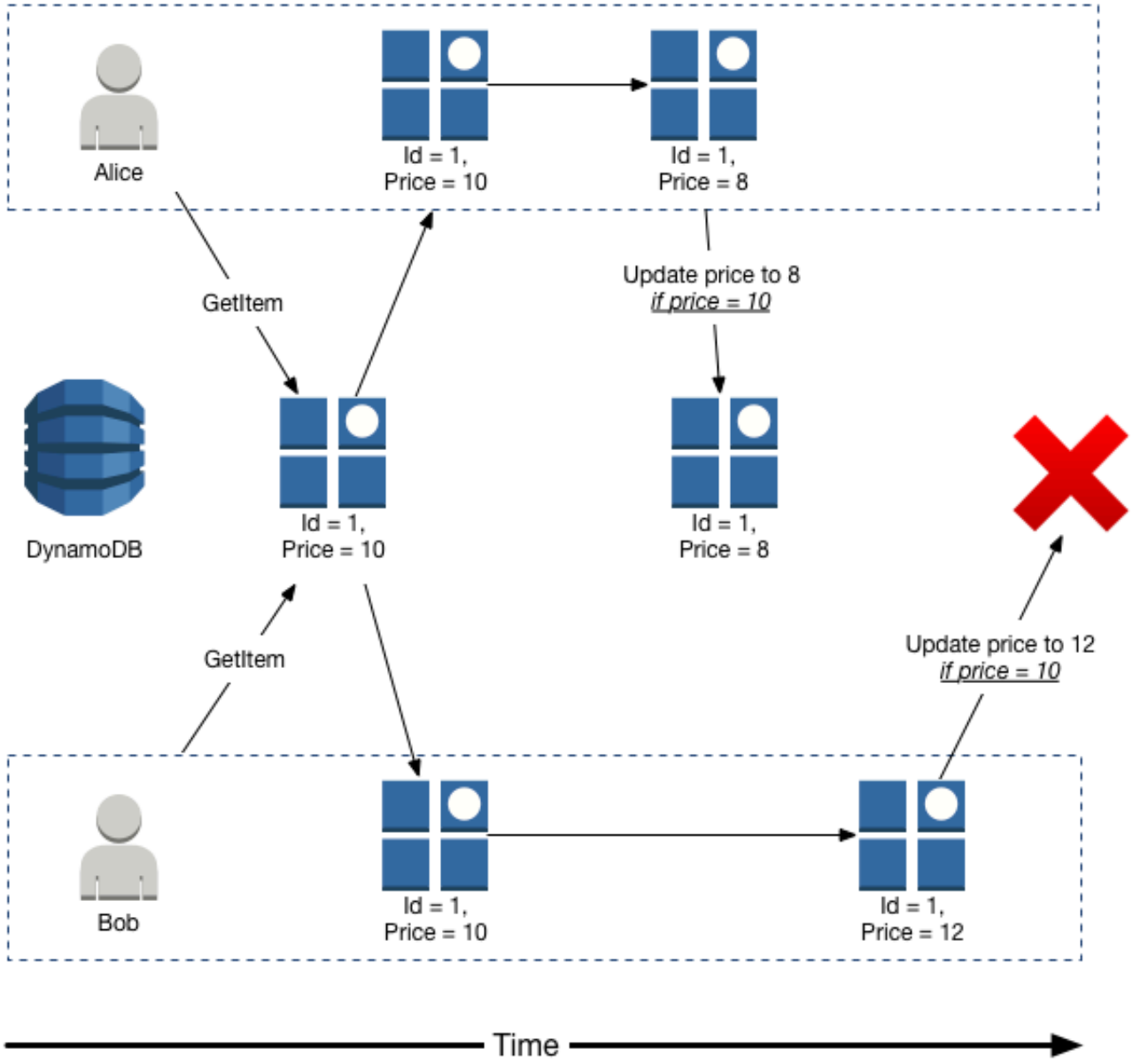
Supposons maintenant que Bob émette une demande `UpdateItem` similaire ultérieurement, mais qu'il redéfinisse l'attribut `Price` sur la valeur 12. Pour Bob, le paramètre `--expression-attribute-values` prend la forme suivante.

```
{
    ":newval":{"N":"12"}
}
```

La demande de Bob aboutit, mais la mise à jour précédente d'Alice est perdue.

Pour demander une opération `PutItem`, `DeleteItem` ou `UpdateItem` conditionnelle, vous spécifiez une expression de condition. Une expression de condition est une chaîne comportant des noms d'attributs, des opérateurs conditionnels et des fonctions intégrées. L'expression entière doit avoir la valeur true. Sinon, l'opération échoue.

Examinez maintenant le diagramme suivant, qui présente les écritures conditionnelles qui empêcheraient le remplacement de la mise à jour d'Alice.



Alice tente dans un premier temps de redéfinir l'attribut `Price` sur la valeur 8, mais seulement si la valeur actuelle de l'attribut `Price` est 10.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --condition-expression "Price = :currval" \
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `expression-attribute-values.json`.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

La mise à jour d'Alice aboutit, car la condition a la valeur true.

Ensuite, Bob tente de redéfinir l'attribut `Price` sur la valeur 12, mais seulement si la valeur actuelle de l'attribut `Price` est 10. Pour Bob, le paramètre `--expression-attribute-values` prend la forme suivante.

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Comme Alice a précédemment redéfinit l'attribut `Price` sur la valeur 8, l'expression de condition a la valeur false et la mise à jour de Bob échoue.

Pour de plus amples informations, veuillez consulter Exemple de commande CLI d'expression de condition DynamoDB.

Idempotence d'écriture conditionnelle

Les écritures conditionnelles peuvent être idempotentes si la vérification conditionnelle porte sur le même attribut que celui mis à jour. Autrement dit, DynamoDB effectue une demande d'écriture donnée seulement si certaines valeurs d'attribut dans l'élément correspondent à ce que vous attendez lors de la demande.

Par exemple, supposons que vous émettiez une demande `UpdateItem` pour augmenter l'attribut `Price` d'un élément de 3, mais seulement si la valeur actuelle de l'attribut `Price` est 20. Après l'envoi de la demande, mais avant que vous n'obteniez les résultats, une erreur réseau se produit et vous ne savez pas si votre demande a abouti ou non. Cette écriture conditionnelle étant idempotente,

vous pouvez retenter la même demande `UpdateItem`, puis DynamoDB met à jour l'élément seulement si l'attribut `Price` est actuellement défini sur la valeur 20.

Unités de capacité consommées par les écritures conditionnelles

Si une `ConditionExpression` prend la valeur false pendant une écriture conditionnelle, DynamoDB continue à consommer la capacité d'écriture de la table. La quantité consommée dépend de la taille de l'élément existant (ou valeur minimale de 1). Par exemple, si un élément existant fait 300 Ko et que l'élément que vous essayez de créer ou mettre à jour fait 310 Ko, les unités de capacité d'écriture consommées seront de 300 Ko si la condition n'est pas remplie et de 310 si elle est remplie. S'il s'agit d'un nouvel élément (élément non existant), les unités de capacité d'écriture consommées seront de 1 si la condition n'est pas remplie et de 310 si elle est remplie.

> **Note**
>
> Les opérations d'écriture consomment uniquement des unités de capacité d'écriture. Elles n'utilisent jamais d'unités de capacité de lecture.

Une écriture conditionnelle ayant échoué renvoie un `ConditionalCheckFailedException`. Lorsque cela se produit, vous ne recevez aucune information dans la réponse concernant la capacité d'écriture consommée.

Renvoyez le nombre d'unités de capacité d'écriture consommées au cours d'une écriture conditionnelle à l'aide du paramètre `ReturnConsumedCapacity` :

- `TOTAL` – Renvoie le nombre total d'unités de capacité d'écriture consommées.
- `INDEXES` – Renvoie le nombre total d'unités de capacité de lecture consommées, avec les sous-totaux correspondant à la table et aux index secondaires concernés par l'opération.
- `NONE` – Aucun détail concernant la capacité d'écriture n'est renvoyé. (Il s'agit de l'option par défaut.)

> **Note**
>
> Contrairement à un index secondaire global, un index secondaire local partage sa capacité de débit alloué avec sa table. L'activité de lecture et d'écriture sur un index secondaire local consomme la capacité de débit alloué de la table.

Utilisation d’expressions dans DynamoDB

Dans Amazon DynamoDB, les expressions vous permettent de spécifier les attributs à lire dans un élément, d’écrire des données lorsqu’une condition est remplie, d’indiquer comment mettre à jour un élément, de définir des requêtes et de filtrer les résultats d’une requête.

Ce tableau décrit la grammaire d’expression basique et les types d’expressions disponibles.

Type d’expression	Description
Expression de projection	Une expression de projection identifie les attributs que vous souhaitez récupérer d'un élément lorsque vous utilisez des opérations telles que GetItem Query ou Scan.
Expression de condition	Une expression de condition détermine quels éléments doivent être modifiés lorsque vous utilisez les DeleteItem opérations PutItem UpdateItem, et.
Expression de mise à jour	Une expression de mise à jour indique comment UpdateItem modifier les attributs d'un élément, par exemple en définissant une valeur scalaire ou en supprimant des éléments d'une liste ou d'une carte.
Expression de condition clé	Une expression de condition clé détermine les éléments qu’une requête doit lire dans une table ou un index.
Expression de filtre	Une expression de filtre détermine les éléments dans les résultats de la requête à vous renvoyer. Tous les autres résultats sont ignorés.

Pour en savoir plus sur la syntaxe d’expression et chaque type d’expression, consultez les sections suivantes.

Rubriques

- Référencement d’attributs d’élément lors de l’utilisation d’expressions dans DynamoDB
- Noms d’attributs d’expression (alias) dans DynamoDB
- Utilisation de valeurs d’attributs d’expression dans DynamoDB
- Utilisation d’expressions de projection dans DynamoDB
- Utilisation d’expressions de mise à jour dans DynamoDB
- Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB
- Exemple de commande CLI d’expression de condition DynamoDB

> **Note**
>
> Pour la compatibilité descendante, DynamoDB prend également en charge des paramètres conditionnels qui n’utilisent pas d’expressions. Pour de plus amples informations, veuillez consulter Paramètres conditionnels hérités de DynamoDB.
> Les nouvelles applications doivent utiliser des expressions plutôt que des paramètres hérités.

Référencement d’attributs d’élément lors de l’utilisation d’expressions dans DynamoDB

Cette section décrit comment référencer des attributs d’élément dans une expression dans Amazon DynamoDB. Vous pouvez utiliser tout attribut, même s’il est profondément imbriqué dans plusieurs listes et mappages.

Rubriques

- Attributs de niveau supérieur
- Attributs imbriqués
- Chemins d’accès à des documents

Un exemple d'article : ProductCatalog

Les exemples sur cette page utilisent l’exemple d’élément suivant dans la table `ProductCatalog`. (Cette table est décrite dans Exemples de tables et de données à utiliser dans DynamoDB.)

```
{
    "Id": 123,
    "Title": "Bicycle 123",
```

```
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
}
```

Notez ce qui suit :

- La valeur de la clé de partition (`Id`) est `123`. Il n'y a aucune clé de tri.
- La plupart des attributs ont des types de données scalaires, comme `String`, `Number`, `Boolean` et `Null`.
- Un attribut (`Color`) est un `String Set`.
- Les attributs suivants sont des types de données de documents :
  - Une liste de `RelatedItems`. Chaque élément est un `Id` pour un produit concerné.
  - Une carte de `Pictures`. Chaque élément est une brève description d'une image, accompagnée d'une URL pour le fichier image correspondant.

- Une carte de `ProductReviews`. Chaque élément représente une évaluation et une liste de commentaires correspondant à cette évaluation. À l'origine, ce mappage est renseigné avec des commentaires cinq étoiles et une étoile.

Attributs de niveau supérieur

Un attribut est dit de niveau supérieur lorsqu'il n'est pas imbriqué dans un autre. Dans le cas de l'élément `ProductCatalog`, voici les attributs de niveau supérieur :

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Tous ces attributs de niveau supérieur sont scalaires, à l'exception de `Color` (liste), `RelatedItems` (liste), `Pictures` (mappage) et `ProductReviews` (mappage).

Attributs imbriqués

Un attribut est dit imbriqué lorsqu'il est incorporé dans un autre. Accédez à un attribut imbriqué à l'aide d'opérateurs de déréférençage :

- `[n]` – Pour des éléments de liste

- `.` (point) – Pour des éléments de mappage

Accès à des éléments de liste

L'opérateur de déréférençage pour un élément de liste est [n], où n est le numéro d'élément. Des éléments de liste sont de base zéro, donc [0] représente le premier élément dans la liste, [1] représente le deuxième et ainsi de suite. Voici quelques exemples :

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

L'élément `ThisList[5]` est lui-même une liste imbriquée. Par conséquent, `ThisList[5][11]` fait référence au douzième élément dans cette liste.

Le nombre entre crochets doit être un nombre entier non négatif. Par conséquent, les expressions suivantes ne sont pas valides :

- `MyList[-1]`
- `MyList[0.4]`

Accès à des éléments de mappage

L'opérateur de déréférençage pour un élément de mappage est . (un point). Utilisez un point comme séparateur entre des éléments d'un mappage :

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Chemins d'accès à des documents

Dans une expression, vous indiquez à DynamoDB où trouver un attribut à l'aide d'un chemin d'accès à un document. Pour un attribut de niveau supérieur, le chemin d'accès au document est simplement le nom d'attribut. Pour un attribut imbriqué, vous construisez le chemin d'accès au document à l'aide d'opérateurs de déréférençage.

Voici quelques exemples d'expressions de chemins d'accès à des documents. (Consultez l'élément présenté dans Référencement d'attributs d'élément lors de l'utilisation d'expressions dans DynamoDB.)

- Attribut scalaire de niveau supérieur.

  `Description`

- Attribut de liste de niveau supérieur. (Renvoie la liste entière, et non uniquement certains éléments.)

  `RelatedItems`

- Le troisième élément de la liste `RelatedItems`. (N'oubliez pas que les éléments de liste sont de base zéro.)

  `RelatedItems[2]`

- Photo de face du produit.

  `Pictures.FrontView`

- Tous les commentaires cinq étoiles.

  `ProductReviews.FiveStar`

- Le premier des commentaires cinq étoiles.

  `ProductReviews.FiveStar[0]`

> **Note**
>
> La profondeur maximale pour un chemin d'accès à un document est de 32. Par conséquent, le nombre d'opérateurs de déréférençage dans un chemin d'accès ne peut pas dépasser cette limite.

Vous pouvez utiliser tout nom d'attribut dans le chemin d'accès à un document, à condition qu'il réponde aux exigences suivantes :

- Le premier caractère est `a-z` ou `A-Z` et/ou `0-9`.
- Le second caractère (le cas échéant) est `a-z`, `A-Z`.

> **Note**
>
> Si un nom d'attribut ne répond pas à cette exigence, vous devez définir un nom d'attribut d'expression comme espace réservé.

Pour de plus amples informations, veuillez consulter Noms d'attributs d'expression (alias) dans DynamoDB.

Noms d'attributs d'expression (alias) dans DynamoDB

Un nom d'attribut d'expression est un alias (ou espace réservé) utilisé dans une expression Amazon DynamoDB à la place d'un nom d'attribut réel. Un nom d'attribut d'expression doit commencer par un signe dièse (#) et être suivi d'un ou de plusieurs caractères alphanumériques. Le trait de soulignement (_) est également autorisé.

Cette section présente plusieurs cas dans lesquels vous devez utiliser des noms d'attributs d'expression.

> **Note**
>
> Les exemples de cette section utilisent le AWS Command Line Interface(AWS CLI).

Rubriques

- Mots réservés
- Noms d'attributs comportant des caractères spéciaux
- Attributs imbriqués
- Référencement répété de noms d'attributs

Mots réservés

Dans certains cas, vous pouvez avoir besoin d'écrire une expression comportant un nom d'attribut qui entre en conflit avec un mot réservé DynamoDB. (Pour obtenir la liste complète des mots réservés, consultez Mots réservés dans DynamoDB.)

Par exemple, l'AWS CLI exemple suivant échouerait car il `COMMENT` s'agit d'un mot réservé.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Pour contourner cela, vous pouvez remplacer `Comment` par un nom d'attribut d'expression tel que `#c`. Le # (signe dièse) est obligatoire et indique qu'il s'agit d'un espace réservé pour un nom d'attribut. L'AWS CLI exemple devrait maintenant ressembler à ce qui suit.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Si un nom d'attribut commence par un nombre ou comporte un espace ou un mot réservé, vous devez remplacer ce nom d'attribut dans l'expression à l'aide d'un nom d'attribut d'expression.

Noms d'attributs comportant des caractères spéciaux

Dans une expression, un point (« . ») est interprété comme un caractère de séparation dans un chemin d'accès à un document. Cependant, DynamoDB vous permet également d'utiliser un caractère point et d'autres caractères spéciaux tels qu'un trait d'union (« - ») dans le cadre d'un nom d'attribut. Cela peut être ambigu dans certains cas. À titre d'illustration, supposons que vous souhaitiez récupérer l'attribut `Safety.Warning` d'un élément `ProductCatalog`. (Consultez Référencement d'attributs d'élément lors de l'utilisation d'expressions dans DynamoDB.)

Supposons que vous souhaitiez accéder à `Safety.Warning` à l'aide d'une expression de projection.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB renvoie un résultat vide au lieu de la chaîne attendue (« `Always wear a helmet` »). En effet, DynamoDB interprète un point dans une expression comme un caractère de séparation dans un chemin d'accès à un document. Dans ce cas, vous devez définir un nom d'attribut d'expression (tel que `#sw`) en remplacement de `Safety.Warning`. Vous pouvez ensuite utiliser l'expression de projection suivante.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB renvoie alors le résultat correct.

> **Note**
>
> Si un nom d'attribut comporte un point (« . ») ou un trait d'union (« - »), vousdevez remplacer le nom de cet attribut dans l'expression à l'aide d'un nom d'attribut d'expression.

Attributs imbriqués

Supposons que vous souhaitiez accéder à l'attribut imbriqué `ProductReviews.OneStar`. Dans un nom d'attribut d'expression, DynamoDB traite un point (« . ») comme un caractère dans un nom d'attribut. Pour référencer l'attribut imbriqué, définissez un nom d'attribut d'expression pour chaque élément dans le chemin d'accès au document :

- `#pr – ProductReviews`
- `#1star – OneStar`

Vous pouvez ensuite utiliser `#pr.#1star` pour l'expression de projection.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB renvoie alors le résultat correct.

Référencement répété de noms d'attributs

Les noms d'attributs d'expression sont utiles lorsque vous devez référencer le même nom d'attribut à plusieurs reprises. Par exemple, imaginons l'expression suivante pour récupérer certains des commentaires d'un élément `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Pour la rendre plus concise, vous pouvez remplacer `ProductReviews` par un nom d'attribut d'expression tel que `#pr`. L'expression révisée ressemblerait maintenant à ceci.

- `#pr.FiveStar, #pr.ThreeStar, #pr.OneStar`

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

Si vous définissez un nom d'attribut d'expression, vous devez l'utiliser systématiquement sur l'ensemble de l'expression. En outre, vous ne pouvez pas ignorer le symbole #.

Utilisation de valeurs d'attributs d'expression dans DynamoDB

Les valeurs d'attributs d'expression dans Amazon DynamoDB font office de variables. Elles remplacent les valeurs réelles que vous souhaitez comparer, valeurs que vous ne connaissez peut-être pas avant l'exécution. Une valeur d'attribut d'expression doit commencer par un signe deux-points (`:`) et être suivie d'un ou de plusieurs caractères alphanumériques.

Par exemple, supposons que vous souhaitiez renvoyer tous les éléments de `ProductCatalog` disponibles en `Black` et coûtant au maximum `500`. Vous pouvez utiliser une opération `Scan` avec une expression de filtre, comme dans cet exemple AWS Command Line Interface (AWS CLI).

```
aws dynamodb scan \
    --table-name ProductCatalog \
```

```
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Si vous définissez une valeur d'attribut d'expression, vous devez l'utiliser systématiquement sur l'ensemble de l'expression. En outre, vous ne pouvez pas ignorer le symbole `:`.

Les valeurs d'attributs d'expression sont utilisées avec des expressions de condition de clé, des expressions de condition, des expressions de mise à jour et des expressions de filtre.

Utilisation d'expressions de projection dans DynamoDB

Vous lisez des données d'une table à l'aide d'opérations telles que `GetItem`, `Query` ou `Scan`. Par défaut, Amazon DynamoDB renvoie tous les attributs d'élément. Obtenez seulement certains des attributs et non leur totalité à l'aide d'une expression de projection.

Une expression de projection est une chaîne qui identifie les attributs souhaités. Pour récupérer un seul attribut, spécifiez son nom. Pour plusieurs attributs, les noms doivent être séparés par des virgules.

Voici des exemples d'expressions de projection basées sur l'élément `ProductCatalog` à partir de Référencement d'attributs d'élément lors de l'utilisation d'expressions dans DynamoDB :

- Un seul attribut de niveau supérieur.

  `Title`

- Trois attributs de niveau supérieur. DynamoDB extrait l'ensemble `Color` tout entier.

  `Title, Price, Color`

- Quatre attributs de niveau supérieur. DynamoDB renvoie tout le contenu de `RelatedItems` et `ProductReviews`.

  `Title, Description, RelatedItems, ProductReviews`

> **Note**
>
> L'expression de projection n'a aucun effet sur la consommation du débit alloué. DynamoDB détermine les unités de capacité consommées en fonction de la taille d'un élément, et non du volume de données renvoyées à une application.

Mots réservés et caractères spéciaux

DynamoDB comporte des mots réservés et des caractères spéciaux. DynamoDB vous permet d'utiliser ces mots réservés et ces caractères spéciaux pour les noms, mais nous vous recommandons d'éviter de le faire, car vous devez utiliser des alias pour eux chaque fois que vous utilisez ces noms dans une expression. Pour obtenir la liste complète, consultez Mots réservés dans DynamoDB.

Vous devez utiliser des noms d'attributs d'expression à la place du nom réel si :

- le nom d'attribut figure dans la liste des mots réservés dans DynamoDB ;
- le nom d'attribut ne répond pas à l'exigence que le premier caractère soit `a-z` ou `A-Z` et le second (le cas échéant) soit `a-Z`, `A-Z` ou `0-9` ;
- le nom d'attribut comporte un signe # (dièse) ou : (deux-points).

L'AWS CLI exemple suivant montre comment utiliser une expression de projection avec une `GetItem` opération. Cette expression de projection récupère un attribut scalaire de niveau supérieur (`Description`), le premier élément d'une liste (`RelatedItems[0]`) et une liste imbriquée dans un mappage (`ProductReviews.FiveStar`).

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

Le code JSON suivant serait renvoyé pour cet exemple.

```
{
    "Item": {
        "Description": {
            "S": "123 description"
        },
```

```
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

Utilisation d'expressions de mise à jour dans DynamoDB

L'opération `UpdateItem` met à jour un élément existant ou ajoute un nouvel élément à la table si celui-ci n'existe pas déjà. Vous devez fournir la clé de l'élément que vous souhaitez mettre à jour. Vous devez également fournir une expression de mise à jour, en indiquant les attributs que vous souhaitez modifier et les valeurs que vous souhaitez leur affecter.

Une expression de mise à jour spécifie comment l'opération `UpdateItem` doit modifier les attributs d'un élément, par exemple en définissant une valeur scalaire ou en supprimant des éléments d'une liste ou d'un mappage.

Voici un résumé de la syntaxe des expressions de mise à jour :

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Une expression de mise à jour se compose d'une ou de plusieurs clauses. Chaque clause commence par un mot clé `SET`, `REMOVE`, `ADD` ou `DELETE`. Vous pouvez inclure l'une de ces clauses dans une expression de mise à jour, dans n'importe quel ordre. Toutefois, chaque mot clé d'action peut n'apparaître qu'une seule fois.

Chaque clause comporte une ou plusieurs actions, séparées par des virgules. Chaque action représente une modification de données.

Les exemples de cette section sont basés sur l'élément `ProductCatalog` présenté dans Utilisation d'expressions de projection dans DynamoDB.

Les rubriques ci-dessous couvrent différents cas d'utilisation de l'action `SET`.

Rubriques

- SET – Modification ou ajout d'attributs d'élément
- REMOVE – Suppression d'attributs d'un élément
- ADD – Mise à jour de nombres et d'ensembles
- DELETE – Suppression d'éléments d'un ensemble
- Utilisation de plusieurs expressions de mise à jour

SET – Modification ou ajout d'attributs d'élément

Ajoutez un ou plusieurs attributs à un élément à l'aide de l'action `SET` dans une expression de mise à jour. Si l'un de ces attributs existe déjà, il est remplacé par les nouvelles valeurs. Si vous souhaitez éviter de remplacer un attribut existant, vous pouvez utiliser l'action `SET` avec la fonction `if_not_exists`. La fonction `if_not_exists` est spécifique à l'action `SET` et utilisable uniquement dans une expression de mise à jour.

Lorsque vous utilisez `SET` pour mettre à jour un élément de la liste, le contenu de cet élément est remplacé par les nouvelles données que vous spécifiez. Si l'élément n'existe pas déjà, `SET` ajoute le nouvel élément à la fin de la liste.

Si vous ajoutez plusieurs éléments dans une seule opération `SET`, les éléments sont triés dans l'ordre par numéro d'élément.

Vous pouvez également effectuer un ajout ou une soustraction à un attribut de type `Number` à l'aide de `SET`. Afin d'effectuer plusieurs actions `SET`, séparez-les par des virgules.

Dans le récapitulatif de la syntaxe suivante :

- L'*path*élément est le chemin du document vers l'élément.
- Un *operand*élément peut être soit le chemin d'un document vers un élément, soit une fonction.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Si l'élément ne contient pas d'attribut au chemin d'accès spécifié, la fonction `if_not_exists` prend la valeur `value`. Dans le cas contraire, elle prend la valeur `path`.

L'opération `PutItem` suivante crée un exemple d'élément auquel nous ferons référence dans les exemples.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Les arguments de la fonction `--item` sont stockés dans le fichier `item.json`. Dans un souci de simplicité, seuls quelques attributs sont utilisés.

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Rubriques

- Modification d'attributs

- Ajout de listes et de mappages
- Ajout d'éléments à une liste
- Ajout d'attributs de mappage imbriqués
- Incrémentation et décrémentation d'attributs numériques
- Ajout d'éléments à une liste
- Empêcher le remplacement d'un attribut existant

Modification d'attributs

Example

Mettez à jour les attributs `ProductCategory` et `Price`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> Dans l'opération `UpdateItem`, `--return-values ALL_NEW` amène DynamoDB à renvoyer l'élément tel qu'il s'affiche après la mise à jour.

Ajout de listes et de mappages

Example

Ajoutez une nouvelle liste et un nouveau mappage.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Ajout d'éléments à une liste

Example

Ajoutez un nouvel attribut à la liste `RelatedItems`. (N'oubliez pas que les éléments de liste sont en base zéro, donc [0] représente le premier élément dans la liste, [1] représente le deuxième et ainsi de suite.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Lorsque vous utilisez SET pour mettre à jour un élément de la liste, le contenu de cet élément est remplacé par les nouvelles données que vous spécifiez. Si l'élément n'existe pas déjà, SET ajoute le nouvel élément à la fin de la liste.
> Si vous ajoutez plusieurs éléments dans une seule opération SET, les éléments sont triés dans l'ordre par numéro d'élément.

Ajout d'attributs de mappage imbriqués

Example

Ajoutez certains attributs de mappage imbriqués.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Les arguments de la fonction `--expression-attribute-names` sont stockés dans le fichier `names.json`.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Important

Vous ne pouvez pas mettre à jour les attributs de carte imbriqués si la carte parent n'existe pas. Si vous tentez de mettre à jour un attribut imbriqué (par exemple,`ProductReviews.FiveStar`) alors que la carte parent (`ProductReviews`) n'existe pas, DynamoDB renvoie `ValidationException` un avec le message « Le chemin du document fourni dans l'expression de mise à jour n'est pas valide pour la mise à jour ». Lorsque vous créez des éléments dont les attributs de carte imbriqués seront mis à jour ultérieurement, initialisez des cartes vides pour les attributs parents. Par exemple :

```
{
    "Id": {"N": "789"},
    "ProductReviews": {"M": {}},
    "Metadata": {"M": {}}
}
```

Cela vous permet de mettre à jour les attributs imbriqués `ProductReviews.FiveStar` sans erreur.

Incrémentation et décrémentation d'attributs numériques

Vous pouvez effectuer un ajout ou une soustraction à un attribut numérique existant. Pour ce faire, utilisez les opérateurs + (plus) et - (moins).

Example

Réduisez l'attribut `Price` d'un élément.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Augmentez l'attribut `Price` à l'aide de l'opérateur + dans l'expression de mise à jour.

Ajout d'éléments à une liste

Vous pouvez ajouter des éléments à la fin d'une liste. Pour ce faire, utilisez `SET` avec la fonction `list_append`. (Le nom de fonction respecte la casse.) La fonction `list_append` est spécifique à l'action `SET` et utilisable uniquement dans une expression de mise à jour. La syntaxe est la suivante.

- `list_append (`*`list1`*`, `*`list2`*`)`

La fonction prend deux listes en entrée et ajoute tous les éléments de *`list2`* à *`list1`*.

Example

Dans Ajout d'éléments à une liste, vous créez la liste `RelatedItems` et la renseignez avec deux éléments : `Hammer` et `Nails`. Vous allez maintenant ajouter deux autres éléments à la fin de `RelatedItems`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
```

```
        ]
    }
}
```

Enfin, vous allez ajouter un autre élément au début de `RelatedItems`. Pour ce faire, permutez l'ordre des éléments `list_append`. (N'oubliez pas que la fonction `list_append` prend deux listes en entrée, puis ajoute la seconde liste à la première.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

L'attribut `RelatedItems` résultant comporte maintenant cinq éléments, dans l'ordre suivant : `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw`.

Empêcher le remplacement d'un attribut existant

Example

Définissez l'attribut `Price` d'un élément, mais seulement si l'élément ne possède pas déjà un attribut `Price`. (Si `Price` existe déjà, il ne se passe rien.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE – Suppression d'attributs d'un élément

Supprimez un ou plusieurs attributs d'un élément dans Amazon DynamoDB à l'aide de l'action `REMOVE` dans une expression de mise à jour. Afin d'effectuer plusieurs actions `REMOVE`, séparez-les par des virgules.

Voici un résumé de la syntaxe pour `REMOVE` dans une expression de mise à jour. Le seul opérande est le chemin d'accès au document pour l'attribut que vous souhaitez supprimer.

```
remove-action ::=
    path
```

Example

Supprimez des attributs d'un élément. (Si les attributs n'existent pas, il ne se passe rien.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Suppression d'éléments d'une liste

Vous pouvez supprimer des éléments d'une liste à l'aide de `REMOVE`.

Example

Dans Ajout d'éléments à une liste, vous avez modifié un attribut de liste (`RelatedItems`) de sorte qu'il comporte les cinq éléments suivants :

- `[0]—Chisel`
- `[1]—Hammer`
- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

L'exemple suivant AWS Command Line Interface(AWS CLI) supprime `Hammer` et `Nails` de la liste.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Après la suppression de `Hammer` et `Nails`, les éléments restants sont déplacés. La liste comporte désormais les éléments suivants :

- `[0]—Chisel`
- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD – Mise à jour de nombres et d'ensembles

> **Note**
>
> En général, nous recommandons d'utiliser `SET` plutôt que de `ADD` garantir des opérations idempotentes.

Ajoutez un nouvel attribut et ses valeurs à un élément à l'aide de l'action `ADD` dans une expression de mise à jour.

Si l'attribut existe déjà, le comportement d'`ADD` dépend du type de données de l'attribut :

- Si l'attribut est un nombre et que la valeur que vous ajoutez est également un nombre, la valeur est ajoutée mathématiquement à l'attribut existant. (Si la valeur est un nombre négatif, elle est soustraite de l'attribut existant.)
- Si l'attribut est un ensemble et que la valeur que vous ajoutez est également un ensemble, la valeur est ajoutée à l'ensemble existant.

> **Note**
>
> L'action `ADD` prend uniquement en charge les données de type nombre et ensemble.

Afin d'effectuer plusieurs actions `ADD`, séparez-les par des virgules.

Dans le récapitulatif de la syntaxe suivante :

- L'*`path`*élément est le chemin du document vers un attribut. L'attribut doit être un `Number` ou un type de données ensemble.
- L'*`value`*élément est un nombre que vous souhaitez ajouter à l'attribut (pour les types de `Number` données) ou un ensemble à ajouter à l'attribut (pour les types d'ensembles).

```
add-action ::=
    path value
```

Les rubriques ci-dessous couvrent différents cas d'utilisation de l'action ADD.

Rubriques

- Ajout d'un nombre
- Ajout d'éléments à un ensemble

Ajout d'un nombre

Supposons que l'attribut `QuantityOnHand` n'existe pas. L'AWS CLI exemple suivant définit `QuantityOnHand` la valeur 5.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD QuantityOnHand :q" \
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

Maintenant que `QuantityOnHand` existe, vous pouvez réexécuter l'exemple pour incrémenter `QuantityOnHand` de 5 chaque fois.

Ajout d'éléments à un ensemble

Supposons que l'attribut `Color` n'existe pas. L'exemple AWS CLI suivant définit `Color` sur un ensemble de chaîne avec deux éléments.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

Maintenant que `Color` existe, vous pouvez lui ajouter de nouveaux éléments.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE – Suppression d'éléments d'un ensemble

> **Important**
>
> L'action `DELETE` prend uniquement en charge les types de données `Set`.

Supprimez un ou plusieurs éléments d'un ensemble à l'aide de l'action `DELETE` dans une expression de mise à jour. Afin d'effectuer plusieurs actions `DELETE`, séparez-les par des virgules.

Dans le récapitulatif de la syntaxe suivante :

- L'*`path`*élément est le chemin du document vers un attribut. L'attribut doit être un type de données ensemble.
- *`subset`*Il s'agit d'un ou de plusieurs éléments que vous souhaitez supprimer*`path`*. Vous devez le spécifier *`subset`* en tant que type d'ensemble.

```
delete-action ::=
    path subset
```

Example

Dans Ajout d'éléments à un ensemble, vous créez l'ensemble de chaînes `Color`. Cet exemple supprime certains des éléments de cet ensemble.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

Utilisation de plusieurs expressions de mise à jour

Vous pouvez utiliser plusieurs expressions de mise à jour dans une seule instruction.

Example

Vous pouvez modifier la valeur d'un attribut et supprimer complètement un autre attribut à l'aide d'une action SET et d'une action REMOVE dans une seule instruction. Cette opération réduirait la valeur de l'attribut `Price` à 15 tout en supprimant l'attribut `InStock` de l'élément.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

Vous pouvez ajouter des éléments à une liste tout en modifiant la valeur d'un autre attribut à l'aide de deux actions SET dans une seule instruction. Cette opération ajouterait « Nails » à l'attribut de liste `RelatedItems` et définirait également la valeur de l'attribut `Price` sur 21.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
 {"N":"21"}}'  \
    --return-values ALL_NEW
```

Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB

Manipulez les données d'une table DynamoDB à l'aide des opérations `PutItem`, `UpdateItem` et `DeleteItem`. Pour ces opérations de manipulation de données, vous pouvez préciser une expression de condition afin de déterminer les éléments à modifier. Si l'expression de condition a la valeur true, l'opération aboutit. Sinon, l'opération échoue.

Cette section traite des fonctions et mots clés intégrés pour l'écriture des expressions de filtre et de condition dans Amazon DynamoDB. Pour en savoir plus sur les fonctions et la programmation avec DynamoDB, consultez Programmation avec DynamoDB et AWS SDKs et le guide de référence de l'API DynamoDB.

Rubriques

- Syntaxe des expressions de filtre et de condition

- Comparaisons
- Fonctions
- Évaluations logiques
- Parenthèses
- Priorité des conditions

Syntaxe des expressions de filtre et de condition

Dans le résumé de syntaxe suivant, un *operand* peut être le suivant :

- Nom d'attribut de niveau supérieur, par exemple `Id`, `Title`, `Description` ou `ProductCategory`
- Chemin d'accès au document référençant un attribut imbriqué

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

Comparaisons

Comparez un opérande à une valeur unique à l'aide de ces comparateurs :

- *a* = *b* — Vrai s'*a*il est égal à*b*.
- *a* <> *b* — Vrai s'il n'*a*est pas égal à*b*.
- *a* < *b* — C'est vrai si *a* c'est inférieur à*b*.
- *a* <= *b* — Vrai s'*a*il est inférieur ou égal à*b*.
- *a* > *b* — Vrai si la valeur *a* est supérieure à*b*.
- *a* >= *b* — Vrai s'*a*il est supérieur ou égal à*b*.

Comparez un opérande à une plage de valeurs ou une liste énumérée de valeurs à l'aide des mots clés `BETWEEN` et `IN` :

- *a* `BETWEEN` *b* `AND` *c* — Vrai s'*a*il est supérieur ou égal à*b*, et inférieur ou égal à*c*.
- *a* `IN (`*b*`, `*c*`, `*d*`)` — Vrai s'il *a* est égal à n'importe quelle valeur de la liste, par exemple, n'importe laquelle de *bc*, ou. *d* La liste peut comporter jusqu'à 100 valeurs séparées par des virgules.

Fonctions

Déterminez si un attribut existe dans un élément ou évaluez la valeur d'un attribut à l'aide des fonctions suivantes. Ces noms de fonction respectent la casse. Pour un attribut imbriqué, vous devez fournir le chemin d'accès au document complet.

Fonction	Description
`attribute_exists (`*path*`)`	True si l'élément comporte l'attribut spécifié par `path`. Exemple : Vérifier si un élément de la table `Product` a une image de vue de côté. • `attribute_exists (#Pictures.#SideView)`

Fonction	Description
`attribute_not_exists (path)`	True si l'attribut spécifié par `path` n'existe pas dans l'élément. Exemple : Vérifier si un élément possède un attribut `Manufacturer` . • `attribute_not_exists (Manufacturer)`

Fonction	Description
`attribute_type ( path, type)`	True si l'attribut à l'emplacement spécifié est d'un type de données particulier. Le paramètre `type` doit avoir l'une des valeurs suivantes : • `S` – String (chaîne) • `SS` – String set (ensemble de chaînes) • `N` – Number (nombre) • `NS` – Number set (ensemble de nombres) • `B` – Binary (binaire) • `BS` – Binary set (ensemble de binaires) • `BOOL` – Boolean (booléen) • `NULL` – Null • `L` – List (liste) • `M` – Map (mappage) Vous devez utiliser une valeur d'attribut d'expression pour le paramètre `type`. Exemple : Vérifier si l'attribut `QuantityOnHand` est de type List (liste). Dans cet exemple, `:v_sub` est un espace réservé pour la chaîne `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)`

Fonction	Description
	Vous devez utiliser une valeur d'attribut d'expression pour le paramètre `type`.
`begins_with ( path, substr)`	True si l'attribut spécifié par `path` commence par une sous-chaîne particulière. Exemple : Vérifier si les tout premiers caractères de l'URL de l'image de vue avant sont `http://`. • `begins_with (Pictures.FrontView, :v_sub)` La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `http://`.

Fonction	Description
`contains (path, operand)`	True si l'attribut spécifié par `path` est l'un des attributs suivants : • Une `String` qui comporte une sous-chaîne particulière • Un `Set` qui comporte un élément particulier au sein de l'ensemble • Une `List` qui comporte un élément particulier Si l'attribut spécifié par `path` est de type `String`, `operand` doit être de type `String`. Si l'attribut spécifié par `path` est de type `Set`, l'`operand` doit être le type d'élément de l'ensemble. Le chemin d'accès et l'opérande doivent être distincts. Autrement dit, `contains (a, a)` renvoie une erreur. Exemple : Vérifier si l'attribut `Brand` comporte la sous-chaîne `Company`. • `contains (Brand, :v_sub)` La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `Company`. Exemple : Vérifier si le produit est disponible en rouge. • `contains (Color, :v_sub)`

Fonction	Description
	La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `Red`.

Fonction	Description
`size (path)`	Renvoie un nombre qui représente la taille d'un attribut. Voici les types de données valides à utiliser avec `size`. Si l'attribut est de type `String`, `size` renvoie la longueur de la chaîne. Exemple : Vérifier si la longueur de la chaîne `Brand` est inférieure ou égale à 20 caractères. La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `20`. • `size (Brand) <= :v_sub` Si l'attribut est de type `Binary`, `size` renvoie le nombre d'octets de la valeur d'attribut. Exemple : supposez que l'élément `ProductCatalog` comporte un binary attribute nommé `VideoClip` qui comporte une courte vidéo du produit utilisé. L'expression suivante vérifie si `VideoClip` dépasse 64 000 octets. La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `64000`. • `size(VideoClip) > :v_sub` Si l'attribut est un type de données `Set`, `size` renvoie le nombre d'éléments dans l'ensemble. Exemple : Vérifier si le produit est disponible dans plusieurs couleurs. La valeur d'attribut

Fonction	Description
	d'expression `:v_sub` est un espace réservé pour `1`. • `size (Color) < :v_sub` Si l'attribut est de type `List` ou `Map`, `size` renvoie le nombre d'éléments enfants. Exemple : Vérifier si le nombre de révisions `OneStar` a dépassé un certain seuil. La valeur d'attribut d'expression `:v_sub` est un espace réservé pour `3`. • `size(ProductReviews.OneStar) > :v_sub`

Évaluations logiques

Effectuez les évaluations logiques à l'aide des mots clés `AND`, `OR` et `NOT`. Dans la liste suivante, *a* et *b* représentez les conditions à évaluer.

- *a* `AND` *b*— C'est vrai si *a* les deux *b* sont vrais.
- *a* `OR` *b*— Vrai si l'un *a* ou l'autre *b* (ou les deux) sont vrais.
- `NOT` *a*— Vrai si *a* c'est faux. Faux si *a* c'est vrai.

Voici un exemple de code avec AND dans une opération.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Parenthèses

Modifiez la priorité d'une évaluation logique à l'aide de parenthèses. Supposons, par exemple, que les conditions *b* 1 *a* et 2 soient vraies et que cette condition *c* soit fausse. L'expression suivante a la valeur true :

- *a* OR *b* AND *c*

Toutefois, si vous placez une condition entre parenthèses, elle est évaluée en premier. Par exemple, l'expression suivante a la valeur false :

- (*a* OR *b*) AND *c*

> **Note**
>
> Vous pouvez imbriquer des parenthèses dans une expression. Les parenthèses les plus intérieures sont évaluées en premier.

Voici un exemple de code avec des parenthèses dans une évaluation logique.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Priorité des conditions

DynamoDB évalue les conditions de gauche à droite à l'aide des règles de priorité suivantes :

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- Parenthèses
- NOT
- AND
- OR

Exemple de commande CLI d'expression de condition DynamoDB

Voici quelques AWS Command Line Interface(AWS CLI) exemples d'utilisation d'expressions de condition. Ces exemples sont basés sur la table `ProductCatalog`, qui a été présentée dans

Référencement d'attributs d'élément lors de l'utilisation d'expressions dans DynamoDB. La clé de partition de cette table est `Id` ; il n'y a aucune clé de tri. L'opération `PutItem` suivante crée un exemple d'élément `ProductCatalog` auquel nous ferons référence dans les exemples.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Les arguments de la fonction `--item` sont stockés dans le fichier `item.json`. Dans un souci de simplicité, seuls quelques attributs sont utilisés.

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Rubriques

- Insertion conditionnelle
- Suppressions conditionnelles
- Mises à jour conditionnelles
- Exemples d'expressions conditionnelles

Insertion conditionnelle

L'opération `PutItem` remplace un élément avec la même clé primaire (le cas échéant). Vous pouvez éviter cela à l'aide d'une expression de condition. Elle permet la poursuite de l'écriture seulement si l'élément en question ne possède pas déjà la même clé primaire.

L'exemple suivant vérifie si la clé primaire existe dans la table à l'aide de la fonction `attribute_not_exists()` avant de tenter l'opération d'écriture.

> **Note**
>
> Si votre clé primaire comprend à la fois une clé de partition (pk) et une clé de tri (sk), le paramètre vérifie si `attribute_not_exists(pk)` ET `attribute_not_exists(sk)` prennent la valeur true ou false avant de tenter l'opération d'écriture.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Si l'expression de condition prend la valeur false, DynamoDB renvoie le message d'erreur suivant : The conditional request failed (Échec de la demande conditionnelle).

> **Note**
>
> Pour en savoir plus sur `attribute_not_exists` et d'autres fonctions, consultez Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB.

Suppressions conditionnelles

Effectuez une suppression conditionnelle à l'aide d'une opération `DeleteItem` avec une expression de condition. L'expression de condition doit avoir la valeur true afin que l'opération aboutisse. Dans le cas contraire, l'opération échoue.

Tenez compte de l'élément défini ci-dessus.

Supposons maintenant que vous souhaitiez supprimer l'élément, mais uniquement dans les conditions suivantes :

- L'attribut `ProductCategory` est soit « Sporting Goods » (Matériels de sport), soit « Gardening Supplies » (Articles de jardinerie).
- La valeur de l'attribut `Price` est comprise entre 500 et 600.

L'exemple suivant tente de supprimer l'élément.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> Dans l'expression de condition, le signe `:` (deux points) indique une valeur d'attribut d'expression, c'est-à-dire un espace réservé pour une valeur réelle. Pour de plus amples informations, veuillez consulter Utilisation de valeurs d'attributs d'expression dans DynamoDB.
> Pour en savoir plus sur `IN`, `AND` et d'autres mots clés, consultez Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB.

Dans cet exemple, la comparaison `ProductCategory` a la valeur true, mais la comparaison `Price` a la valeur false. Par conséquent, l'expression de condition prend la valeur false et l'opération `DeleteItem` échoue.

Mises à jour conditionnelles

Effectuez une mise à jour conditionnelle à l'aide d'une opération `UpdateItem` avec une expression de condition. L'expression de condition doit avoir la valeur true afin que l'opération aboutisse. Dans le cas contraire, l'opération échoue.

> **Note**
>
> `UpdateItem` prend également en charge les expressions de mise à jour, dans lesquelles vous spécifiez les modifications à apporter à un élément. Pour de plus amples informations, veuillez consulter Utilisation d'expressions de mise à jour dans DynamoDB.

Supposons que vous commenciez avec l'élément défini ci-dessus.

L'exemple suivant effectue une opération `UpdateItem`. Elle tente de réduire la valeur de l'attribut `Price` d'un produit de 75, mais l'expression de condition empêche la mise à jour si la valeur actuelle de l'attribut `Price` est inférieure à 500.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Si la valeur de départ de l'attribut `Price` est 650, l'opération `UpdateItem` réduit la valeur de l'attribut `Price` à 575. Si vous réexécutez l'opération `UpdateItem`, la valeur de l'attribut `Price` est réduite à 500. Si vous l'exécutez une troisième fois, l'expression de condition prend la valeur false et l'opération de mise à jour échoue.

> **Note**
>
> Dans l'expression de condition, le signe `:` (deux points) indique une valeur d'attribut d'expression, c'est-à-dire un espace réservé pour une valeur réelle. Pour de plus amples informations, veuillez consulter Utilisation de valeurs d'attributs d'expression dans DynamoDB.
> Pour en savoir plus sur « > » et d'autres opérateurs, consultez Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB.

Exemples d'expressions conditionnelles

Pour en savoir plus sur les fonctions utilisées dans les exemples suivants, consultez Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB. Pour en savoir plus sur la spécification de différents types d'attributs dans une expression, consultez Référencement d'attributs d'élément lors de l'utilisation d'expressions dans DynamoDB.

Vérification des attributs d'un élément

Vous pouvez vérifier l'existence (ou la non-existence) d'un attribut. Si l'expression de condition a la valeur true, l'opération aboutit. Dans le cas contraire, l'opération échoue.

L'exemple suivant supprime un produit seulement s'il ne possède pas d'attribut `Price` à l'aide de `attribute_not_exists`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB fournit également une fonction `attribute_exists`. L'exemple suivant supprime un produit seulement si les commentaires le concernant sont négatifs.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Vérification du type d'attribut

Vous pouvez vérifier le type de données d'une valeur d'attribut à l'aide de la fonction `attribute_type`. Si l'expression de condition a la valeur true, l'opération aboutit. Dans le cas contraire, l'opération échoue.

L'exemple suivant supprime un produit seulement s'il possède un attribut `Color` de type String Set à l'aide de `attribute_type`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments pour `--expression-attribute-values` sont stockés dans le fichier expression-attribute-values .json.

```
{
```

```
    ":v_sub":{"S":"SS"}
}
```

Vérification de la valeur de début de la chaîne

Vous pouvez vérifier si une valeur d'attribut String commence par une sous-chaîne spécifique à l'aide de la fonction `begins_with`. Si l'expression de condition a la valeur true, l'opération aboutit. Dans le cas contraire, l'opération échoue.

L'exemple suivant supprime un produit seulement si l'élément `FrontView` de la carte `Pictures` commence par une valeur spécifique à l'aide de `begins_with`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments pour `--expression-attribute-values` sont stockés dans le fichier expression-attribute-values .json.

```
{
    ":v_sub":{"S":"http://"}
}
```

Recherche d'un élément dans un ensemble

Vous pouvez rechercher un élément dans un ensemble ou rechercher une sous-chaîne dans une chaîne à l'aide de la fonction `contains`. Si l'expression de condition a la valeur true, l'opération aboutit. Dans le cas contraire, l'opération échoue.

L'exemple suivant supprime un produit seulement si l'attribut de type String Set `Color` comporte un élément avec une valeur spécifique à l'aide de `contains`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments pour `--expression-attribute-values` sont stockés dans le fichier expression-attribute-values .json.

```
{
    ":v_sub":{"S":"Red"}
}
```

Vérification de la taille d'une valeur d'attribut

Vous pouvez vérifier la taille d'une valeur d'attribut à l'aide de la fonction `size`. Si l'expression de condition a la valeur true, l'opération aboutit. Dans le cas contraire, l'opération échoue.

L'exemple suivant supprime un produit seulement si la taille du binary attribute `VideoClip` est supérieure à `64000` octets à l'aide de `size`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

Les arguments pour `--expression-attribute-values` sont stockés dans le fichier expression-attribute-values .json.

```
{
    ":v_sub":{"N":"64000"}
}
```

Utilisation de la durée de vie (TTL) dans DynamoDB

La durée de vie (TTL) de DynamoDB est une méthode économique pour supprimer les éléments qui ne sont plus pertinents. Elle vous permet de définir un horodatage d'expiration par élément pour indiquer quand cet élément n'est plus nécessaire. DynamoDB supprime automatiquement les éléments ayant expiré quelques jours après leur date d'expiration, sans consommer de débit d'écriture.

Pour utiliser la TTL, activez-la d'abord sur une table, puis définissez un attribut spécifique pour stocker l'horodatage d'expiration de la TTL. L'horodatage doit être stocké au format d'heure epoch Unix avec une granularité de quelques secondes. Chaque fois qu'un élément est créé ou mis à jour, vous pouvez calculer le délai d'expiration et l'enregistrer dans l'attribut TTL.

Les éléments dont les attributs TTL sont valides et ayant expiré peuvent être supprimés par le système à tout moment, généralement quelques jours après leur expiration. Vous pouvez à tout moment mettre à jour les éléments ayant expiré qui sont en attente de suppression, notamment en modifiant ou en supprimant leurs attributs TTL. Lors de la mise à jour d'un élément ayant expiré, nous vous recommandons d'utiliser une expression conditionnelle pour vous assurer que l'élément n'a pas été supprimé ultérieurement. Utilisez des expressions de filtre pour supprimer les éléments ayant expiré des résultats Scan et Query.

Les éléments supprimés fonctionnent de la même manière que ceux supprimés par le biais d'opérations de suppression classiques. Une fois supprimés, les éléments sont placés dans DynamoDB Streams sous forme de suppression de service au lieu d'être supprimés par l'utilisateur, et sont supprimés des index secondaires locaux et globaux comme avec les autres opérations de suppression.

Si vous utilisez des tables globales version 2019.11.21 (actuelle) et que vous utilisez également la fonctionnalité TTL, DynamoDB réplique les suppressions par TTL sur toutes les tables de réplica. La suppression par TTL initiale ne consomme pas d'unités de capacité d'écriture (WCU) dans la région dans laquelle l'expiration de la TTL a lieu. Toutefois, la suppression par TTL répliquée dans les tables de réplica consomme une unité de capacité d'écriture répliquée lorsque vous utilisez la capacité provisionnée ou une unité d'écriture répliquée lorsque vous utilisez le mode de capacité à la demande, dans chacune des régions de réplica, et des frais s'appliquent.

Pour plus d'informations sur TTL, veuillez consulter les rubriques suivantes :

Rubriques

- Activation de la durée de vie (TTL) dans DynamoDB
- Calcul de la durée de vie (TTL) dans DynamoDB
- Utilisation des éléments ayant expiré et de la durée de vie (TTL)

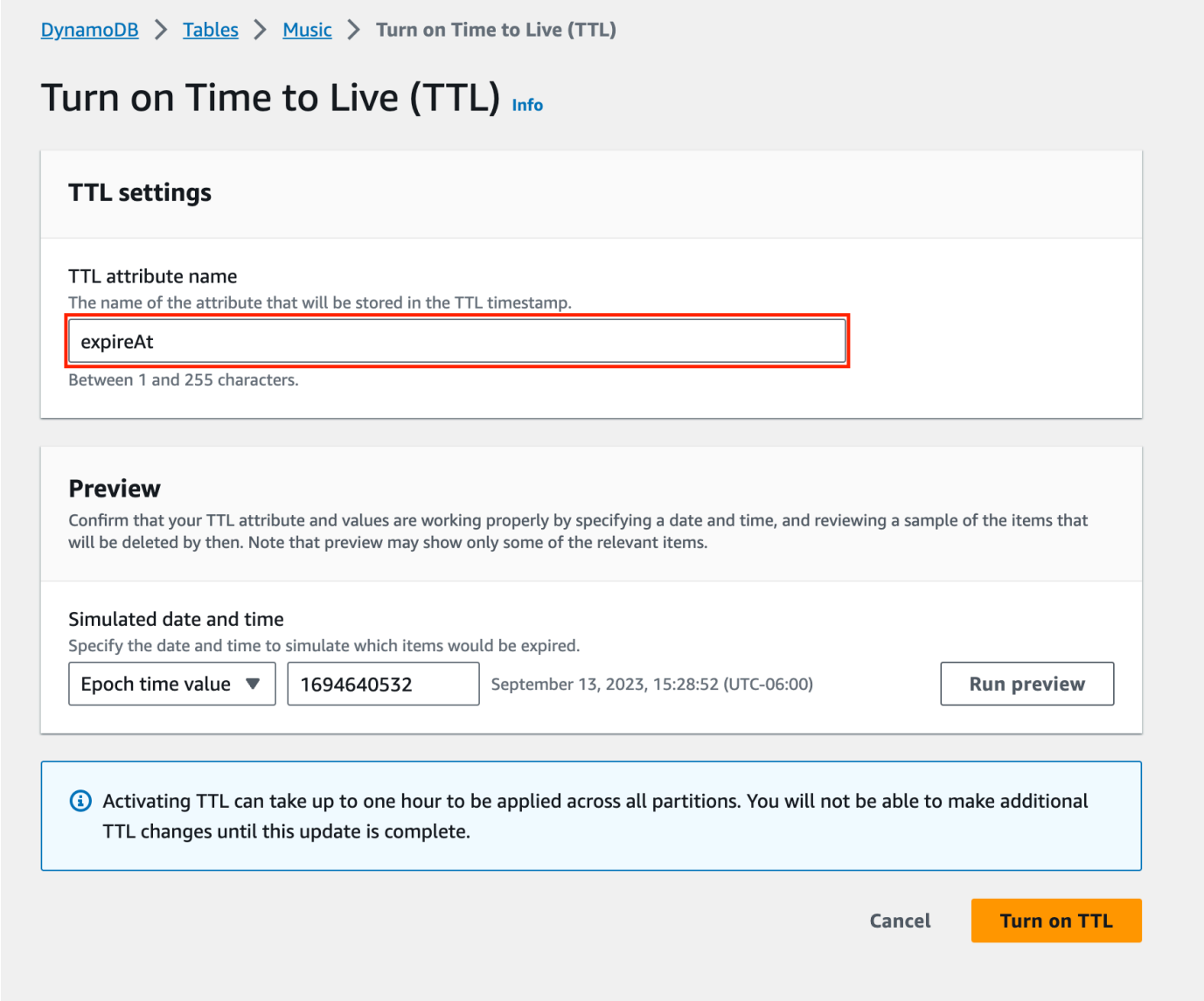
Activation de la durée de vie (TTL) dans DynamoDB

> **Note**
>
> Pour faciliter le débogage et la vérification de la fonctionnalité TTL, les valeurs fournies pour la TTL de l'élément sont journalisées en texte brut dans les journaux de diagnostic DynamoDB.

Vous pouvez activer la TTL dans la console Amazon DynamoDB, dans l'AWS Command Line Interface (AWS CLI) ou en utilisant la Référence de l'API Amazon DynamoDB avec l'un des kits AWS SDK prévus. L'activation de la durée de vie (TTL) sur toutes les partitions prend environ une heure.

Activation de la TTL DynamoDB à l'aide de la console AWS

1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Choisissez Tables, puis choisissez la table à modifier.
3. Dans l'onglet Paramètres supplémentaires, dans la section Durée de vie (TTL), choisissez Activer.
4. Lorsque vous activez TTL sur une table, vous devez identifier un nom d'attribut spécifique que le service DynamoDB recherchera pour déterminer si un élément peut faire l'objet d'une expiration. Le nom de l'attribut TTL, illustré ci-dessous, distingue les majuscules et minuscules et doit correspondre à l'attribut défini dans vos opérations de lecture et d'écriture. En cas de non-concordance, les éléments ayant expiré ne seront pas supprimés. Pour renommer l'attribut TTL, vous devez le désactiver, puis le réactiver avec le nouveau nom. La TTL continue à traiter les suppressions pendant environ 30 minutes après sa désactivation. Elle doit être reconfigurée sur les tables restaurées.



5. (Facultatif) Vous pouvez effectuer un test en simulant la date et l'heure de l'expiration et en faisant correspondre quelques éléments. Cela vous fournit un exemple de liste d'éléments et confirme qu'il existe des éléments contenant le nom d'attribut TTL fourni avec la date d'expiration.

Une fois que la TTL est activée, l'attribut TTL est marqué TTL lorsque vous affichez les éléments dans la console DynamoDB. Vous pouvez afficher la date et l'heure d'expiration d'un élément en faisant passer votre souris au-dessus de l'attribut.

Activation de la TTL DynamoDB à l'aide de l'API

Python

Vous pouvez activer la TTL avec du code à l'aide de l'opération UpdateTimeToLive.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )

        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
```

```
enable_ttl('your-table-name', 'expirationDate')
```

Vous pouvez confirmer que la TTL est activée à l'aide de l'opération DescribeTimeToLive, qui décrit l'état TTL d'une table. L'état `TimeToLive` est `ENABLED` ou `DISABLED`.

```
# create a DynamoDB client
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

JavaScript

Vous pouvez activer la TTL avec du code à l'aide de l'opération UpdateTimeToLiveCommand.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
```

```
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Activation de la durée de vie à l'aide de l'AWS CLI

1. Active TTL sur la table `TTLExample`.

   ```
   aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
   specification "Enabled=true, AttributeName=ttl"
   ```

2. Décrit TTL sur la table `TTLExample`.

   ```
   aws dynamodb describe-time-to-live --table-name TTLExample
   {
       "TimeToLiveDescription": {
           "AttributeName": "ttl",
           "TimeToLiveStatus": "ENABLED"
       }
   }
   ```

3. Ajoutez un élément à la table `TTLExample` avec l'attribut de TTL défini à l'aide du shell BASH et de l'AWS CLI.

   ```
   EXP=`date -d '+5 days' +%s`
   aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
    {"N": "'$EXP'"}}'
   ```

Cet exemple démarre par la date actuelle à laquelle cinq jours sont ajoutés pour créer une date d'expiration. Il a ensuite converti l'heure d'expiration en heure au format epoch pour ajouter en dernier lieu un élément à la table « `TTLExample` ».

> **Note**
>
> Pour définir des valeurs d'expiration pour TTL, l'une des méthodes à votre disposition consiste à calculer le nombre de secondes à ajouter à l'heure d'expiration. Par exemple, 5

jours équivalent à 432 000 secondes. Toutefois, il est souvent préférable de commencer par une date et de définir le reste en fonction de celle-ci.

Il est relativement simple d'obtenir l'heure actuelle au format epoch, comme dans l'exemple suivant.

- Terminal Linux : `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Activation de la TTL DynamoDB à l'aide d'CloudFormation

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TTLExampleTable:
    Type: AWS::DynamoDB::Table
    Description: "A DynamoDB table with TTL Specification enabled"
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
      TimeToLiveSpecification:
        AttributeName: "TTLExampleAttribute"
        Enabled: true
```

Pour plus d'informations sur l'utilisation de la TTL dans vos modèles CloudFormation, cliquez ici.

Calcul de la durée de vie (TTL) dans DynamoDB

Une méthode courante pour implémenter la durée de vie, ou TTL, consiste à définir un délai d'expiration pour les éléments en fonction de leur date de création ou de dernière mise à jour. Cela peut être fait en ajoutant l'heure aux horodatages `createdAt` et `updatedAt`. Par exemple, la TTL des éléments qui viennent d'être créés peut être défini sur `createdAt` + 90 jours. Lorsque l'élément est mis à jour, la TTL peut être recalculée sur `updatedAt` + 90 jours.

Le délai d'expiration calculé doit être au format epoch, en secondes. Pour que l'expiration et la suppression soient prises en compte, la TTL ne doit pas dater de plus de cinq ans. Si vous utilisez un autre format, les processus TTL ignorent l'élément. Si vous définissez la date d'expiration à une date future à laquelle vous souhaitez que l'élément expire, l'élément expirera après cette date. Supposons, par exemple, que vous ayez défini la date d'expiration sur 1724241326 (qui correspond au lundi 21 août 2024 11:55:26 (GMT)). L'élément expirera après l'heure spécifiée.

Rubriques

- Création d'un élément et définition de la durée de vie (TTL)
- Mise à jour d'un élément et actualisation de la durée de vie (TTL)

Création d'un élément et définition de la durée de vie (TTL)

L'exemple suivant montre comment calculer le délai d'expiration lors de la création d'un élément, en utilisant `expireAt` comme nom d'attribut TTL. Une instruction d'affectation obtient l'heure actuelle sous forme de variable. Dans l'exemple, le délai d'expiration est calculé comme étant de 90 jours à compter de l'heure actuelle. L'heure est ensuite convertie au format epoch et enregistrée sous forme de type de données « entier » dans l'attribut TTL.

Les exemples de code suivants montrent comment créer un élément avec une TTL.

Java

SDK pour Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
```

```
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
```

```
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
```

```
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Pour plus de détails sur l'API, voir PutItemla section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);
```

```
    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Pour plus de détails sur l'API, voir PutItemla section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e
```

```
# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Pour plus de détails sur l'API, consultez PutItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Mise à jour d'un élément et actualisation de la durée de vie (TTL)

Cet exemple est la suite de la section précédente. Le délai d'expiration peut être recalculé si l'élément est mis à jour. L'exemple suivant recalcule l'horodatage `expireAt` pour qu'il corresponde à 90 jours à partir de l'heure actuelle.

Les exemples de code suivants montrent comment mettre à jour la TTL d'un élément.

Java

SDK pour Java 2.x

Mise à jour de la TTL sur un élément DynamoDB existant dans une table.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;
```

```
        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API, voir UpdateItemla section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}
```

```
// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Pour plus de détails sur l'API, voir UpdateItemla section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
```

```
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Les exemples de TTL présentés dans cette introduction illustrent une méthode permettant de garantir que seuls les éléments récemment mis à jour sont conservés dans une table. Les éléments mis à jour voient leur durée de vie prolongée, tandis que les éléments non mis à jour après leur création expirent et sont supprimés gratuitement, ce qui réduit le stockage utilisé et permet de maintenir les tables propres.

Utilisation des éléments ayant expiré et de la durée de vie (TTL)

Les éléments ayant expiré qui sont en attente de suppression peuvent être exclus des opérations de lecture et d'écriture. Cela est utile dans les scénarios où les données ayant expiré ne sont plus valides et ne doivent pas être utilisées. S'ils ne sont pas exclus, ils continueront à s'afficher dans les opérations de lecture et d'écriture jusqu'à ce qu'ils soient supprimés par le processus en arrière-plan.

> **Note**
>
> Ces éléments sont toujours pris en compte dans les coûts de stockage et de lecture jusqu'à ce qu'ils soient supprimés.

Les suppressions par TTL peuvent être identifiées dans DynamoDB Streams, mais uniquement dans la région où la suppression a eu lieu. Les suppressions par TTL qui sont répliquées dans les régions

de tables globales ne sont pas identifiables dans les flux DynamoDB des régions vers lesquelles la suppression est répliquée.

Exclusion des éléments ayant expiré des opérations de lecture

Pour les opérations de lecture telles que Scan et Query, une expression de filtre peut exclure les éléments ayant expiré qui sont en attente de suppression. Comme indiqué dans l'extrait de code suivant, l'expression de filtre peut exclure les éléments dont la durée de vie, ou TTL, est inférieure ou égale à l'heure actuelle. Par exemple, le code du kit SDK Python inclut une instruction d'affectation qui obtient l'heure actuelle sous forme de variable (now) et la convertit en `int` au format d'heure epoch.

Les exemples de code suivants montrent l'interrogation d'éléments TTL.

Java

SDK pour Java 2.x

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
```

```
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK pour JavaScript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
```

```
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK pour Python (Boto3)

```
from datetime import datetime

import boto3
```

```
def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Écriture conditionnelle sur les éléments ayant expiré

Une expression conditionnelle peut être utilisée pour éviter les écritures sur les éléments ayant expiré. L'extrait de code ci-dessous est une mise à jour conditionnelle qui vérifie si le délai d'expiration est supérieur à l'heure actuelle. Si tel est le cas, l'opération d'écriture se poursuit.

Les exemples de code suivants montrent comment mettre à jour la TTL d'un élément de manière conditionnelle.

Java

SDK pour Java 2.x

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
```

```
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
 PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
 TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
 failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
        this.dynamoDbClient = null;
    }

    /**
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
```

```
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);
```

```
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
```

```
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
```

```
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
```

```
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Identification des éléments supprimés dans DynamoDB Streams

L'enregistrement de flux contient un champ d'identité utilisateur `Records[<index>].userIdentity`. Les éléments qui sont supprimés par le processus TTL ont les champs suivants :

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

L'extrait JSON suivant montre la partie pertinente d'un seul enregistrement de flux :

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Interrogation de tables dans DynamoDB

Vous pouvez utiliser l'opération d'API `Query` dans Amazon DynamoDB pour rechercher des éléments en fonction de valeurs de clé primaire.

Vous devez fournir le nom de l'attribut de clé de partition et une valeur unique pour cet attribut. `Query` retourne tous les éléments avec cette valeur de clé de partition. Vous pouvez le cas échéant fournir un attribut de clé de tri et utiliser un opérateur de comparaison pour affiner les résultats de recherche.

Pour plus d'informations sur l'utilisation`Query`, tels que la syntaxe de requête, les paramètres de réponse et d'autres exemples, voirInterrogationdans leAmazon DynamoDB API Reference.

Rubriques

- Expressions de condition clé pour l'opération Query dans DynamoDB
- Expressions de filtre pour l'opération Query dans DynamoDB
- Pagination des résultats de requête de table dans DynamoDB
- Autres aspects de l'utilisation de l'opération Query dans DynamoDB

Expressions de condition clé pour l'opération Query dans DynamoDB

Vous pouvez utiliser n'importe quel nom dans une expression de condition de clé, sous réserve que le premier caractère soit `a-z` ou `A-Z` et que les autres caractères (à compter du deuxième caractère s'il y en a un) soit `a-z`, `A-Z` ou `0-9`. En outre, le nom d'attribut ne doit pas être un mot réservé DynamoDB. (Pour en obtenir la liste complète, consultez Mots réservés dans DynamoDB.) Si un nom d'attribut ne répond pas à ces exigences, vous devez définir un nom d'attribut d'expression comme espace réservé. Pour en savoir plus, consultez Noms d'attributs d'expression (alias) dans DynamoDB.

Pour les éléments avec une valeur de clé de partition donnée, DynamoDB stocke ces éléments proches les uns des autres, triés par valeur de clé de tri. Dans une opération `Query`, DynamoDB récupère les éléments en ordre trié, puis traite les éléments à l'aide d'une `KeyConditionExpression` et d'une `FilterExpression` éventuellement présente. Alors seulement les résultats de `Query` sont renvoyés au client.

Une opération `Query` retourne toujours un ensemble de résultats. Si aucun élément correspondant n'est trouvé, le jeu de résultats est vide.

Les résultats de `Query` sont toujours triés sur la valeur de la clé de tri. Si le type de données de la clé de tri est `Number`, les résultats sont retournés par ordre numérique. Sinon, les résultats sont retournés dans l'ordre des octets UTF-8. Par défaut, l'ordre de tri est croissant. Pour inverser l'ordre, définissez le paramètre `ScanIndexForward` sur `false`.

Une seule opération `Query` peut extraire au maximum 1 Mo de données. Cette limite s'applique avant qu'une `FilterExpression` ou `ProjectionExpression` ne soit appliquée aux résultats. Si `LastEvaluatedKey` est présent dans la réponse et n'a pas la valeur null, vous devez paginer le jeu de résultats (voir Pagination des résultats de requête de table dans DynamoDB).

Exemples d'expression de condition clé

Pour spécifier les critères de recherche, vous utilisez une expression de condition de clé, c'est-à-dire une chaîne qui détermine les éléments à lire dans la table ou l'index.

Vous devez spécifier le nom de la clé de partition et la valeur comme condition d'égalité. Vous ne pouvez pas utiliser d'attribut non-clé dans une expression de condition clé.

Le cas échéant, vous pouvez fournir une deuxième condition pour la clé de tri (si elle est présente). La condition de la clé de tri doit être l'un des opérateurs de comparaison suivants :

- *a* = *b* – True si l'attribut *a* est égal à la valeur *b*.
- *a* < *b* – True si *a* est inférieur à *b*.
- *a* <= *b* – True si *a* est inférieur ou égal à *b*.
- *a* > *b* – True si *a* est supérieur à *b*.
- *a* >= *b* – True si *a* est supérieur ou égal à *b*.
- *a* `BETWEEN` *b* `AND` *c* – True si *a* est supérieur ou égal à *b*, et inférieur ou égal à *c*.

La fonction suivante est également prise en charge :

- `begins_with (`*a*`, `*substr*`)` – True si la valeur de l'attribut *a* commence par une sous-chaîne particulière.

Les exemples AWS Command Line Interface (AWS CLI) suivants illustrent l'utilisation d'expressions de condition de clé. Ces expressions utilisent des espaces réservés (tels que `:name` et `:sub`) au lieu de valeurs réelles. Pour plus d'informations, consultez Noms d'attributs d'expression (alias) dans DynamoDB et Utilisation de valeurs d'attributs d'expression dans DynamoDB.

Example

Interrogez la table `Thread` pour un `ForumName` (clé de partition) particulier. Tous les éléments ayant cette valeur `ForumName` sont lus par la requête, car la clé de tri (`Subject`) n'est pas incluse dans `KeyConditionExpression`.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Interrogez la table `Thread` en quête d'un `ForumName` (clé de partition) particulier, mais cette fois ne retournez que les éléments avec un `Subject` (clé de tri) donné.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Interrogez la table `Reply` en quête d'un `Id` (clé de partition) particulier, mais retournez uniquement les éléments dont `ReplyDateTime` (clé de tri) commence par certains caractères.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
```

```
}
```

Expressions de filtre pour l'opération Query dans DynamoDB

Si vous avez besoin d'affiner davantage les résultats de l'opération `Query`, vous pouvez fournir une expression de filtre facultative. Une expression de filtre détermine les éléments dans les résultats de l'opération `Query` qui doivent vous être renvoyés. Tous les autres résultats sont ignorés.

Une expression de filtre est appliquée après la fin de l'opération `Query`, mais avant que les résultats soient renvoyés. Par conséquent, une opération `Query` utilise la même capacité de lecture, qu'une expression de filtre soit présente ou non.

Une opération `Query` peut extraire au maximum 1 Mo de données. Cette limite s'applique avant que l'expression de filtre soit évaluée.

Une expression de filtre ne peut pas contenir d'attributs de clé de partition ou de clé de tri. Vous devez préciser ces attributs dans l'expression de condition de clé, pas dans l'expression de filtre.

La syntaxe d'un expression de filtre est similaire à celle d'une expression de condition de clé. Les expressions de filtre peuvent utiliser les mêmes comparateurs, fonctions et opérateurs logiques qu'une expression de condition de clé. En outre, les expressions de filtre peuvent utiliser l'opérateur non égal (<>), l'opérateur `OR`, l'opérateur `CONTAINS`, l'opérateur `IN`, l'opérateur `BEGINS_WITH`, l'opérateur `BETWEEN`, l'opérateur `EXISTS` et l'opérateur `SIZE`. Pour en savoir plus, consultez Expressions de condition clé pour l'opération Query dans DynamoDB et Syntaxe des expressions de filtre et de condition.

Example

L'exemple AWS CLI suivant interroge la table `Thread` en quête d'un `ForumName` (clé de partition) particulier et d'un `Subject` (clé de tri). Parmi les éléments trouvés, seules les unités d'exécution de discussion les plus populaires sont renvoyés, c'est-à-dire les threads ayant plus qu'un certain nombre de `Views`.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject begins_with :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

Les arguments de la fonction `--expression-attribute-values` sont stockés dans le fichier `values.json`.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Notez que `Views` étant un mot réservé dans DynamoDB (consultez Mots réservés dans DynamoDB), cet exemple utilise `#v` comme espace réservé. Pour en savoir plus, consultez Noms d'attributs d'expression (alias) dans DynamoDB.

> **Note**
>
> Une expression de filtre supprime des éléments du jeu de résultats de `Query`. Dans la mesure du possible, évitez d'utiliser `Query` lorsque vous pensez extraire un grand nombre d'éléments mais que vous devrez en supprimer une grande partie.

Pagination des résultats de requête de table dans DynamoDB

DynamoDB pagine les résultats des opérations `Query`. Avec la pagination, les résultats `Query` sont répartis en « pages » de données d'une taille maximum de 1 Mo. Une application peut traiter la première page des résultats, puis la deuxième, et ainsi de suite.

Une opération `Query` retourne uniquement un ensemble de résultats correspondant à la limite de taille de 1 Mo. Pour déterminer si le nombre de résultats est plus important et pour récupérer une page à la fois, les applications doivent procéder de la manière suivante :

1. Examinez le résultat de l'opération `Query` de niveau inférieur :
   - Si le résultat contient un élément `LastEvaluatedKey` dont la valeur n'est pas null, passez à l'étape 2.
   - S'il n'y a pas de `LastEvaluatedKey` dans le résultat, il n'y a plus aucun élément à récupérer.
2. Construisez une `Query` avec la même `KeyConditionExpression`. Cependant, cette fois, acceptez la valeur `LastEvaluatedKey` de l'étape 1 et utilisez-la comme paramètre `ExclusiveStartKey` dans la nouvelle demande `Query`.
3. Exécutez la nouvelle demande `Query`.

4. Passez à l'étape 1.

En d'autres termes, l'élément `LastEvaluatedKey` provenant d'une réponse `Query` doit être utilisé comme `ExclusiveStartKey` pour la demande `Query` suivante. Si aucun élément `LastEvaluatedKey` n'est présent dans la réponse `Query`, vous avez extrait la dernière page de résultats. Si `LastEvaluatedKey` n'est pas vide, cela ne signifie pas nécessairement qu'il y a plus de données dans le jeu de résultats. Le seul moyen de savoir que vous avez atteint la fin du jeu de résultats est lorsque `LastEvaluatedKey` est vide.

Vous pouvez afficher ce comportement à l'aide de l'AWS CLI. L'AWS CLI envoie des demandes `Query` de bas niveau à DynamoDB de manière répétée jusqu'à ce que `LastEvaluatedKey` ne figure plus dans les résultats. Étudiez l'exemple d'AWS CLI suivant qui récupère les titres de film d'une année donnée.

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

D'ordinaire, AWS CLI gère la pagination automatiquement. Cependant, dans cet exemple, le paramètre AWS CLI `--page-size` limite le nombre d'éléments par page. La paramètre `--debug` imprime les informations de bas niveau relatives aux demandes et aux réponses.

Si vous exécutez l'exemple, la première réponse de DynamoDB est à similaire à ce qui suit.

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

Le `LastEvaluatedKey` de la réponse indique que certains éléments n'ont pas été récupérés. L'AWS CLI émet alors une autre demande `Query` à DynamoDB. Ce modèle de requête et de réponse se poursuit jusqu'à l'obtention d'une réponse finale.

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
```

```
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

L'absence de `LastEvaluatedKey` indique qu'il n'y a pas plus de d'élément à récupérer.

> **Note**
>
> Les kits SDK AWS manipulent les réponses de DynamoDB de bas niveau (y compris la présence ou l'absence de `LastEvaluatedKey`), et fournissent diverses abstractions pour la pagination des résultats de `Query`. Par exemple, l'interface de document du kit SDK pour Java fournit le support `java.util.Iterator` pour vous permettre de parcourir les résultats un par un.
> Pour des exemples de code dans divers langages de programmation, consultez le Guide de prise en main d'Amazon DynamoDB et la documentation du kit SDK AWS pour votre langage.

Vous pouvez également réduire la taille de la page en limitant le nombre d'éléments dans l'ensemble de résultats, avec le paramètre `Limit` de l'opération `Query`.

Pour plus d'informations sur l'interrogation avec DynamoDB, consultez Interrogation de tables dans DynamoDB.

Autres aspects de l'utilisation de l'opération Query dans DynamoDB

Cette section couvre d'autres aspects de l'opération Query DynamoDB, notamment la limitation de la taille des résultats, le comptage des éléments analysés par rapport aux éléments renvoyés, la surveillance de la consommation de capacité de lecture et le contrôle de la cohérence en lecture.

Limiter le nombre d'éléments dans le jeu de résultats

Avec l'opération `Query`, vous pouvez limiter le nombre d'éléments qu'elle lit. Pour ce faire, définissez le paramètre `Limit` sur le nombre maximal d'éléments souhaité.

Par exemple, supposons que vous effectuiez une opération `Query` sur une table, avec une valeur `Limit` de 6 et sans expression de filtre. Le résultat `Query` contient les six premiers éléments de la table qui correspondent à l'expression de condition de clé de la demande.

Supposons maintenant que vous ajoutiez une expression de filtre à l'opération `Query`. Dans ce cas, DynamoDB lit jusqu'à six éléments, puis renvoie uniquement ceux qui correspondent à l'expression

de filtre. Le résultat `Query` final contient au maximum six éléments, même si davantage d'éléments auraient correspondu à l'expression de filtre si DynamoDB avait continué à lire plus d'éléments.

Comptabilisation des éléments dans les résultats

Outre les éléments qui correspondent à vos critères, la réponse d'une opération `Query` contient les éléments suivants :

- `ScannedCount` – Nombre d'éléments qui correspondaient à l'expression de condition de clé avant qu'une expression de filtre (le cas échéant) soit appliquée.
- `Count` – Nombre d'éléments qui restent après l'application d'une expression de filtre (le cas échéant).

> **Note**
>
> Si vous n'utilisez pas d'expression de filtre, alors `ScannedCount` et `Count` ont la même valeur.

Si la taille de l'ensemble de résultats `Query` est supérieure à 1 Mo, les opérations `ScannedCount` et `Count` représentent seulement un compte partiel du total des éléments. Vous devez effectuer plusieurs opérations `Query` pour pouvoir extraire tous les résultats (consultez Pagination des résultats de requête de table dans DynamoDB).

Chaque réponse `Query` comporte les `ScannedCount` et `Count` des éléments traités par cette demande `Query` particulière. Pour obtenir les totaux de toutes les demandes `Query`, vous pouvez garder un compte actif de `ScannedCount` et de `Count`.

Unités de capacité consommées par la requête

Vous pouvez interroger n'importe quelle table ou n'importe quel index secondaire avec `Query`, à condition de fournir le nom de l'attribut de clé de partition et une valeur unique pour cet attribut. `Query` renvoie tous les éléments contenant cette valeur de clé de partition. Vous pouvez éventuellement fournir un attribut de clé de tri et utiliser un opérateur de comparaison pour affiner les résultats de recherche. Les opérations d'API `Query` consomment des unités de capacité de lecture, comme suit.

Si vous effectuez une opération **`Query`** sur…	DynamoDB consomme des unités de capacité de lecture de…
Table	Capacité de lecture allouée de la table.
GSI	Capacité de lecture allouée à l'index.
Index secondaire local	Capacité de lecture allouée de la table de base.

Par défaut, l'opération `Query` ne renvoie pas de données concernant la consommation de capacité de lecture. Vous pouvez toutefois spécifier le paramètre `ReturnConsumedCapacity` dans une demande `Query` pour obtenir ces informations. Voici les paramètres valides pour `ReturnConsumedCapacity` :

- `NONE` – Aucune donnée de capacité consommée n'est renvoyée. (Il s'agit de l'option par défaut.)
- `TOTAL` – La réponse inclut le nombre agrégé d'unités de capacité de lecture consommées.
- `INDEXES` – La réponse indique le nombre agrégé d'unités de capacité de lecture consommées, ainsi que la capacité consommée pour chaque table et index consultés.

DynamoDB calcule le nombre d'unités de capacité de lecture consommées en fonction du nombre d'éléments et de la taille de ces éléments, et non du volume de données renvoyées à une application. Pour cette raison, le nombre d'unités de capacité consommées sera le même que vous demandiez tous les attributs (comportement par défaut) ou seulement certains d'entre eux (avec une expression de projection). Le nombre est également le même, que vous utilisiez ou non une expression de filtre. `Query` consomme une unité de capacité de lecture minimale pour effectuer une lecture fortement cohérente par seconde ou deux lectures cohérentes à terme par seconde pour un élément d'une taille maximale de 4 Ko. Si vous devez lire un élément d'une taille supérieure à 4 Ko, DynamoDB a besoin d'unités de demande de lecture supplémentaires. Les tables vides et les très grandes tables comportant un nombre restreint de clés de partition peuvent entraîner la facturation de certaines RCU supplémentaires au-delà du volume de données interrogées. Cela couvre les frais de traitement de la demande `Query`, même en l'absence de données.

Cohérence en lecture de la requête

Par défaut, une opération `Query` effectue des lectures cohérentes à terme. Cela signifie que les résultats de l'opération `Query` peuvent ne pas refléter des modifications apportées par des

opérations `PutItem` ou `UpdateItem` récentes. Pour en savoir plus, consultez Cohérence en lecture DynamoDB.

Si vous avez besoin de lectures fortement cohérentes, définissez le paramètre `ConsistentRead` sur `true` dans la demande `Query`.

Analyse de tables dans DynamoDB

Une opération `Scan` dans Amazon DynamoDB lit tous les éléments d'une table ou d'un index secondaire. Par défaut, une opération `Scan` renvoie tous les attributs de données pour chaque élément de la table ou de l'index. Vous pouvez utiliser le paramètre `ProjectionExpression` de sorte que `Scan` renvoie uniquement certains des attributs, plutôt que leur totalité.

`Scan` renvoie toujours un ensemble de résultats. Si aucun élément correspondant n'est trouvé, l'ensemble de résultats est vide.

Une seule demande `Scan` permet d'extraire un maximum de 1 Mo de données. Le cas échéant, DynamoDB peut appliquer une expression de filtre à ces données, en affinant les résultats avant qu'ils soient renvoyés à l'utilisateur.

Rubriques

- Filtrer les expressions à des fins d'analyse
- Limiter le nombre d'éléments dans l'ensemble de résultats
- Pagination des résultats
- Comptabilisation des éléments dans les résultats
- Unités de capacité consommées par l'opération d'analyse
- Cohérence en lecture de l'opération d'analyse
- Analyse parallèle

Filtrer les expressions à des fins d'analyse

Si vous devez affiner davantage les résultats de l'opération `Scan`, vous pouvez également fournir une expression de filtre. Une expression de filtre détermine les éléments dans les résultats de l'opération `Scan` qui doivent vous être renvoyés. Tous les autres résultats sont ignorés.

Une expression de filtre est appliquée après la fin de l'opération `Scan`, mais avant que les résultats soient renvoyés. Par conséquent, une opération `Scan` consomme la même capacité de lecture, qu'une expression de filtre soit présente ou non.

Une opération `Scan` permet d'extraire au maximum 1 Mo de données. Cette limite s'applique avant que l'expression de filtre soit évaluée.

Avec `Scan`, vous pouvez spécifier tous les attributs dans une expression de filtre, y compris les attributs de clé de partition et de clé de tri.

La syntaxe d'un expression de filtre est identique à celle d'une expression de condition. Les expressions de filtre peuvent utiliser les mêmes comparateurs, fonctions et opérateurs logiques qu'une expression de condition. Pour en savoir plus sur les opérateurs, consultez Expressions de condition et de filtre, opérateurs et fonctions dans DynamoDB.

Example

L'exemple suivant AWS Command Line Interface(AWS CLI) analyse le `Thread` tableau et ne renvoie que les derniers éléments publiés par un utilisateur en particulier.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Limiter le nombre d'éléments dans l'ensemble de résultats

L'opération `Scan` permet de limiter le nombre d'éléments renvoyés dans le résultat. Pour ce faire, définissez le paramètre `Limit` sur le nombre maximal d'éléments que vous souhaitez que l'opération `Scan` renvoie, avant l'évaluation de l'expression de filtre.

Par exemple, supposons que vous effectuiez une opération `Scan` sur une table, avec une valeur `Limit` de 6 et sans expression de filtre. Le résultat de l'opération `Scan` comporte les six premiers éléments de la table.

Supposons maintenant que vous ajoutiez une expression de filtre à l'opération `Scan`. Dans ce cas, DynamoDB applique l'expression de filtre aux six éléments renvoyés et supprime ceux qui ne correspondent pas. Le résultat final de l'opération `Scan` comporte six éléments au plus, selon le nombre d'éléments filtrés.

Pagination des résultats

DynamoDB pagine les résultats des opérations `Scan`. Avec la pagination, les résultats de l'opération `Scan` sont répartis en « pages » de données d'une taille maximum de 1 Mo. Une application peut traiter la première page des résultats, puis la deuxième, et ainsi de suite.

Une opération `Scan` renvoie uniquement un ensemble de résultats correspondant à la limite de taille de 1 Mo.

Pour déterminer si le nombre de résultats est plus important et récupérer une page à la fois, les applications doivent procéder comme suit :

1. Examinez le résultat de l'opération `Scan` de niveau inférieur :
   - Si le résultat comporte un élément `LastEvaluatedKey`, passez à l'étape 2.
   - S'il n'y a pas de `LastEvaluatedKey` dans le résultat, il n'y a plus aucun élément à récupérer.
2. Construisez une nouvelle demande `Scan`, avec les mêmes paramètres que la précédente. Cependant, cette fois-ci, acceptez la valeur `LastEvaluatedKey` de l'étape 1 et utilisez-la comme paramètre `ExclusiveStartKey` dans la nouvelle demande `Scan`.
3. Exécutez la nouvelle demande `Scan`.
4. Passez à l'étape 1.

Autrement dit, l'élément `LastEvaluatedKey` provenant d'une réponse de l'opération `Scan` doit être utilisé comme `ExclusiveStartKey` pour la demande `Scan` suivante. Si aucun élément `LastEvaluatedKey` n'est présent dans la réponse de l'opération `Scan`, vous avez extrait la dernière page de résultats. (L'absence de `LastEvaluatedKey` est le seul moyen de savoir que vous avez atteint la fin de l'ensemble de résultats.)

Vous pouvez utiliser le AWS CLI pour visualiser ce comportement.AWS CLI envoie des `Scan` requêtes de bas niveau à DynamoDB, à plusieurs reprises, `LastEvaluatedKey` jusqu'à ce qu'elles ne soient plus présentes dans les résultats. Prenons l'AWS CLI exemple suivant qui scanne l'intégralité du `Movies` tableau mais ne renvoie que les films d'un genre particulier.

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

Normalement, il AWS CLI gère automatiquement la pagination. Toutefois, dans cet exemple, le AWS CLI`--page-size` paramètre limite le nombre d'éléments par page. La paramètre `--debug` imprime les informations de bas niveau relatives aux demandes et aux réponses.

> **Note**
>
> Vos résultats de pagination diffèrent également en fonction des paramètres d'entrée que vous transmettez.
>
> - L'utilisation de `aws dynamodb scan --table-name Prices --max-items 1` renvoie un `NextToken`
> - L'utilisation de `aws dynamodb scan --table-name Prices --limit 1` renvoie un `LastEvaluatedKey`.
>
> Sachez également que l'utilisation de `--starting-token` en particulier requiert la valeur `NextToken`.

Si vous exécutez l'exemple, la première réponse de DynamoDB est similaire à ce qui suit.

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

Le `LastEvaluatedKey` de la réponse indique que certains éléments n'ont pas été récupérés. Il envoie AWS CLI ensuite une autre `Scan` demande à DynamoDB. Ce modèle de requête et de réponse se poursuit jusqu'à l'obtention d'une réponse finale.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

L'absence de `LastEvaluatedKey` indique qu'il n'y a pas plus de d'élément à récupérer.

> **Note**
>
> Ils AWS SDKs gèrent les réponses DynamoDB de bas niveau (y compris la présence ou l'absence de) et fournissent diverses abstractions pour `LastEvaluatedKey` la pagination

des résultats. `Scan` Par exemple, l'interface de document du kit SDK pour Java fournit le support `java.util.Iterator` pour vous permettre de parcourir les résultats un par un. Pour des exemples de code dans divers langages de programmation, consultez le Guide de prise en main d'Amazon DynamoDB et la documentation du kit SDK AWS pour votre langage.

Comptabilisation des éléments dans les résultats

Outre les éléments qui correspondent à vos critères, la réponse d'une opération `Scan` comporte les éléments suivants :

- `ScannedCount` – Nombre d'éléments évalués avant l'application de `ScanFilter`. Une valeur `ScannedCount` élevée avec un nombre faible ou nul de résultats `Count` indique une opération `Scan` inefficace. Si vous n'avez pas utilisé de filtre dans la demande, `ScannedCount` et `Count` ont la même valeur.
- `Count` – Nombre d'éléments qui restent après l'application d'une expression de filtre (le cas échéant).

> **Note**
>
> Si vous n'utilisez pas d'expression de filtre, `ScannedCount` et `Count` ont la même valeur.

Si la taille de l'ensemble de résultats `Scan` est supérieure à 1 Mo, les opérations `ScannedCount` et `Count` représentent seulement un décompte partiel du nombre total d'éléments. Vous devez effectuer plusieurs opérations `Scan` pour extraire tous les résultats. (Consultez Pagination des résultats.)

Chaque réponse `Scan` comporte les `ScannedCount` et `Count` des éléments traités par cette demande `Scan` particulière. Pour obtenir les totaux de toutes les demandes `Scan`, vous pouvez garder un compte actif de `ScannedCount` et `Count`.

Unités de capacité consommées par l'opération d'analyse

Vous pouvez effectuer une opération `Scan` sur toute table ou tout index secondaire. Les opérations `Scan` consomment des unités de capacité de lecture, comme suit.

Si vous effectuez une opération **`Scan`** sur…	DynamoDB consomme des unités de capacité de lecture de…
Table	Capacité de lecture allouée de la table.
GSI	Capacité de lecture allouée à l'index.
Index secondaire local	Capacité de lecture allouée de la table de base.

> **Note**
>
> L'accès intercompte pour les opérations d'analyse d'index secondaires n'est actuellement pas pris en charge par les politiques basées sur les ressources.

Par défaut, une opération `Scan` ne renvoie pas de données concernant la consommation de capacité de lecture. Vous pouvez toutefois spécifier le paramètre `ReturnConsumedCapacity` dans une demande `Scan` pour obtenir ces informations. Voici les paramètres valides pour `ReturnConsumedCapacity` :

- `NONE` – Aucune donnée de capacité consommée n'est renvoyée. (Il s'agit de l'option par défaut.)
- `TOTAL` – La réponse inclut le nombre agrégé d'unités de capacité de lecture consommées.
- `INDEXES` – La réponse indique le nombre agrégé d'unités de capacité de lecture consommées, ainsi que la capacité consommée pour chaque table et index consultés.

DynamoDB calcule le nombre d'unités de capacité de lecture consommées en fonction du nombre d'éléments et de la taille de ces éléments, et non du volume de données renvoyées à une application. Pour cette raison, le nombre d'unités de capacité consommées sera le même que vous demandiez tous les attributs (comportement par défaut) ou seulement certains d'entre eux (avec une expression de projection). Le nombre est également le même, que vous utilisiez ou non une expression de filtre. `Scan` consomme une unité de capacité de lecture minimale pour effectuer une lecture fortement cohérente par seconde ou deux lectures cohérentes à terme par seconde pour un élément d'une taille maximale de 4 Ko. Si vous devez lire un élément d'une taille supérieure à 4 Ko, DynamoDB a besoin d'unités de demande de lecture supplémentaires. Les tables vides et les très grandes tables contenant peu de clés de partition peuvent entraîner des RCUs frais supplémentaires au-delà de la

quantité de données numérisées. Cela couvre les frais de traitement de la demande `Scan`, même en l'absence de données.

Cohérence en lecture de l'opération d'analyse

Par défaut, une opération `Scan` effectue des lectures cohérentes à terme. Autrement dit, les résultats de l'opération `Scan` peuvent ne pas refléter des modifications apportées par des opérations `PutItem` ou `UpdateItem` récentes. Pour de plus amples informations, veuillez consulter Cohérence en lecture DynamoDB.

Si vous avez besoin de lectures fortement cohérentes au moment où l'opération `Scan` commence, définissez le paramètre `ConsistentRead` sur la valeur `true` dans la demande `Scan`. Ainsi, toutes les opérations d'écriture terminées avant le début de l'opération `Scan` sont incluses dans la réponse `Scan`.

La définition de `ConsistentRead` sur `true` peut être utile dans la sauvegarde des tables ou les scénarios de réplication, conjointement avec DynamoDB Streams. Vous utilisez d'abord `Scan` avec le paramètre `ConsistentRead` défini sur la valeur true pour obtenir une copie cohérente des données de la table. Pendant l'opération `Scan`, DynamoDB Streams enregistre toute activité d'écriture supplémentaire qui se produit sur la table. Une fois l'opération `Scan` terminée, vous pouvez appliquer l'activité d'écriture du flux vers la table.

> **Note**
>
> Notez qu'une opération `Scan` avec le paramètre `ConsistentRead` défini sur la valeur `true` consomme deux fois plus d'unités de capacité de lecture, par rapport au fait de laisser la paramètre `ConsistentRead` défini sur sa valeur par défaut (`false`).

Analyse parallèle

Par défaut, l'opération `Scan` traite les données de manière séquentielle. Amazon DynamoDB renvoie des données à l'application par incréments de 1 Mo, et une application effectue des opérations `Scan` supplémentaires pour extraire le Mo de données suivant.

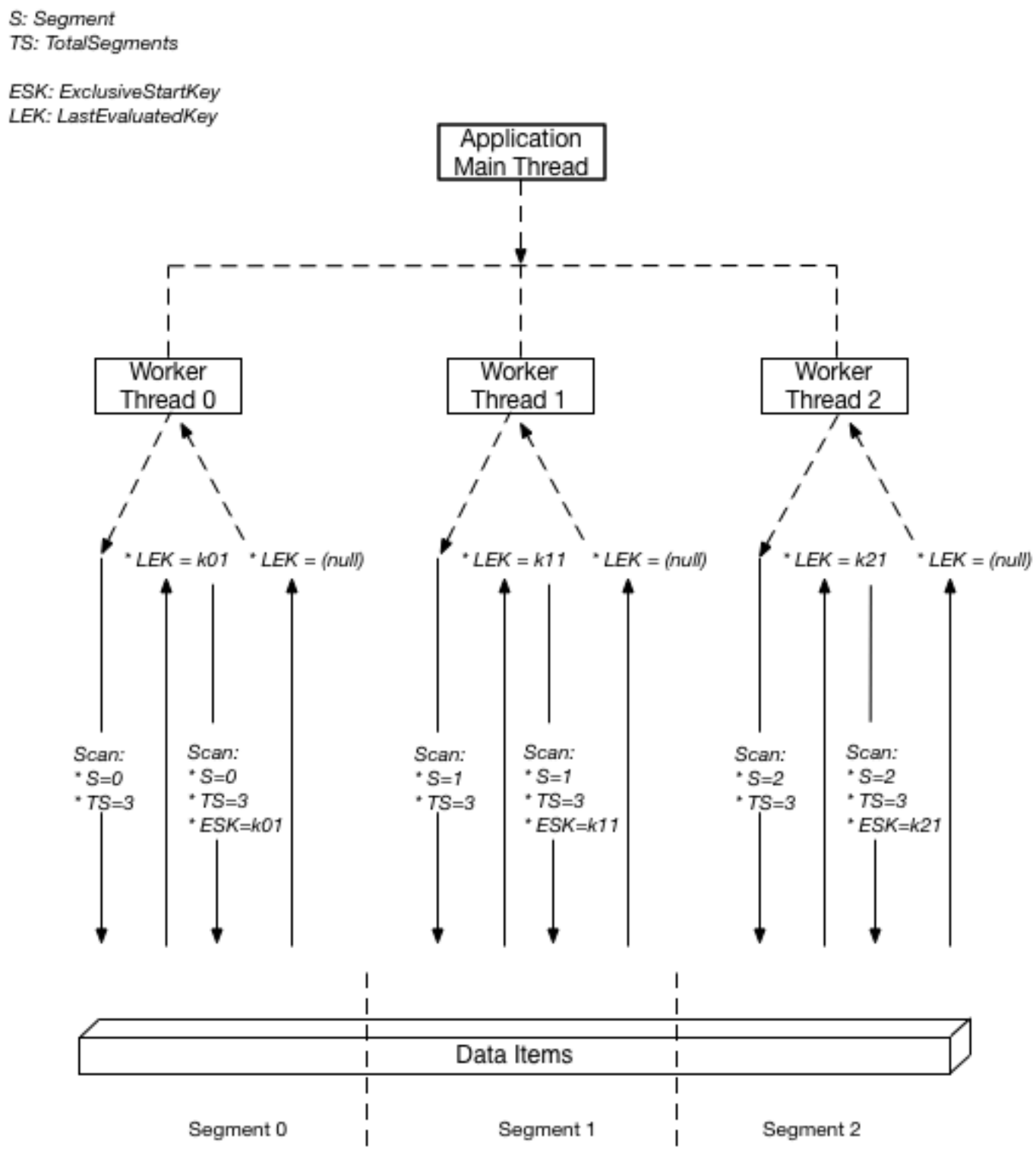
Plus la table ou l'index en cours d'analyse est grand(e), plus l'opération `Scan` prend du temps. En outre, il arrive qu'une opération `Scan` séquentielle ne puisse pas utiliser pleinement la capacité de débit de lecture alloué. Même si DynamoDB répartit les données d'une grande table sur plusieurs

partitions physiques, une opération `Scan` ne peut lire qu'une seule partition à la fois. C'est pourquoi le débit d'une opération `Scan` est limité par le débit maximum d'une partition.

Pour résoudre ce problème, l'opération `Scan` peut diviser logiquement une table ou un index secondaire en plusieurs segments, avec plusieurs workers d'application qui analysent les segments en parallèle. Chaque worker peut être un thread (lorsque le langage de programmation prend en charge le multithreading) ou un processus de système d'exploitation. Pour effectuer une analyse en parallèle, chaque worker émet sa propre demande `Scan` avec les paramètres suivants :

- `Segment` – Segment à analyser par un worker particulier. Chaque worker doit utiliser une valeur différente pour `Segment`.
- `TotalSegments` – Nombre total de segments pour l'analyse en parallèle. Cette valeur doit être identique au nombre de workers que votre application va utiliser.

Le diagramme suivant illustre la manière dont une application multithread exécute une opération `Scan` en parallèle avec trois degrés de parallélisme.



Dans ce diagramme, l'application produit trois threads auxquels elle affecte un numéro. (Les segments étant de base zéro, le premier numéro est toujours 0.) Chaque thread émet une demande `Scan`, définissant le paramètre `Segment` sur son numéro désigné, et le paramètre `TotalSegments`

sur 3. Chaque thread analyse son segment désigné en extrayant 1 Mo de données à la fois, puis renvoie les données au thread principal de l'application.

DynamoDB affecte des éléments à des segments en appliquant une fonction de hachage à la clé de partition de chaque élément. Pour une `TotalSegments` valeur donnée, tous les éléments ayant la même clé de partition sont toujours assignés à la même valeur`Segment`. Cela signifie que dans un tableau où les éléments 1, 2 et 3 se partagent tous `pk="account#123"` (mais ont des clés de tri différentes), ces éléments seront traités par le même opérateur, quelles que soient les valeurs des clés de tri ou la taille de la collection d'articles.

Comme l'attribution des segments est basée uniquement sur le hachage de la clé de partition, les segments peuvent être répartis de manière inégale. Certains segments peuvent ne contenir aucun élément, tandis que d'autres peuvent contenir de nombreuses clés de partition avec de grandes collections d'éléments. Par conséquent, l'augmentation du nombre total de segments ne garantit pas des performances d'analyse plus rapides, en particulier lorsque les clés de partition ne sont pas réparties uniformément dans l'espace de touches.

Les valeurs des paramètres `Segment` et `TotalSegments` s'appliquent à des demandes `Scan` individuelles et vous pouvez utiliser différentes valeurs à tout moment. Il se peut que vous deviez tester ces valeurs et le nombre de workers que vous utilisez, jusqu'à ce que votre application atteigne des performances optimales.

> **Note**
>
> Une analyse en parallèle avec de nombreux threads peut facilement consommer tout le débit alloué pour la table ou l'index analysé(e). Il est préférable d'éviter de telles analyses si d'autres applications soumettent également la table ou l'index à une intense activité de lecture ou d'écriture.
>
> Ccontrôlez le volume de données renvoyées par demande à l'aide du paramètre `Limit`. Cela vous permet d'éviter des situations où un worker consomme tout le débit alloué, au détriment des autres.

PartiQL – Langage de requête compatible SQL pour Amazon DynamoDB

Amazon DynamoDB prend en charge PartiQL, un langage de requête compatible SQL, pour sélectionner, insérer, mettre à jour et supprimer des données dans Amazon DynamoDB. Grâce à PartiQL, vous pouvez facilement interagir avec les tables DynamoDB et exécuter des requêtes ad

hoc à l'aide de AWS Management Console NoSQL Workbench et de DynamoDB pour PartiQL.AWS Command Line Interface APIs

Les opérations PartiQL offrent une disponibilité, une latence et des performances identiques aux autres opérations de plan de données DynamoDB.

Les sections suivantes décrivent l'implémentation DynamoDB de PartiQL.

Rubriques

- Qu'est-ce que PartiQL ?
- PartiQL dans Amazon DynamoDB
- Mise en route avec PartiQL pour DynamoDB
- Types de données PartiQL pour DynamoDB
- Instructions PartiQL pour DynamoDB
- Utilisation de fonctions PartiQL avec DynamoDB
- Opérateurs arithmétiques, de comparaison et logiques PartiQL pour DynamoDB
- Exécution de transactions avec PartiQL pour DynamoDB
- Exécution d'opérations par lot avec PartiQL pour DynamoDB
- Politiques de sécurité IAM avec PartiQL pour DynamoDB

Qu'est-ce que PartiQL ?

Le langage PartiQL fournit un accès aux requêtes compatible SQL sur plusieurs magasins de données contenant des données structurées, des données semi-structurées et des données imbriquées. Il est largement utilisé au sein d'Amazon et est désormais disponible dans le cadre de nombreux AWS services, dont DynamoDB.

Pour la spécification de PartiQL et un didacticiel sur le langage de requête de base, consultez la Documentation PartiQL.

> **Note**
>
> - Amazon DynamoDB prend en charge un sous-ensemble du langage de requête PartiQL.
> - Amazon DynamoDB ne prend pas en charge le format de données Amazon Ion ou les littéraux Amazon Ion.

PartiQL dans Amazon DynamoDB

Pour exécuter des requêtes PartiQL dans DynamoDB, vous pouvez utiliser les ressources suivantes :

- La console DynamoDB
- Le NoSQL Workbench
- Le AWS Command Line Interface(AWS CLI)
- Le DynamoDB APIs

Pour plus d'informations sur l'utilisation de ces méthodes pour accéder à DynamoDB, consultez Accès à DynamoDB.

Mise en route avec PartiQL pour DynamoDB

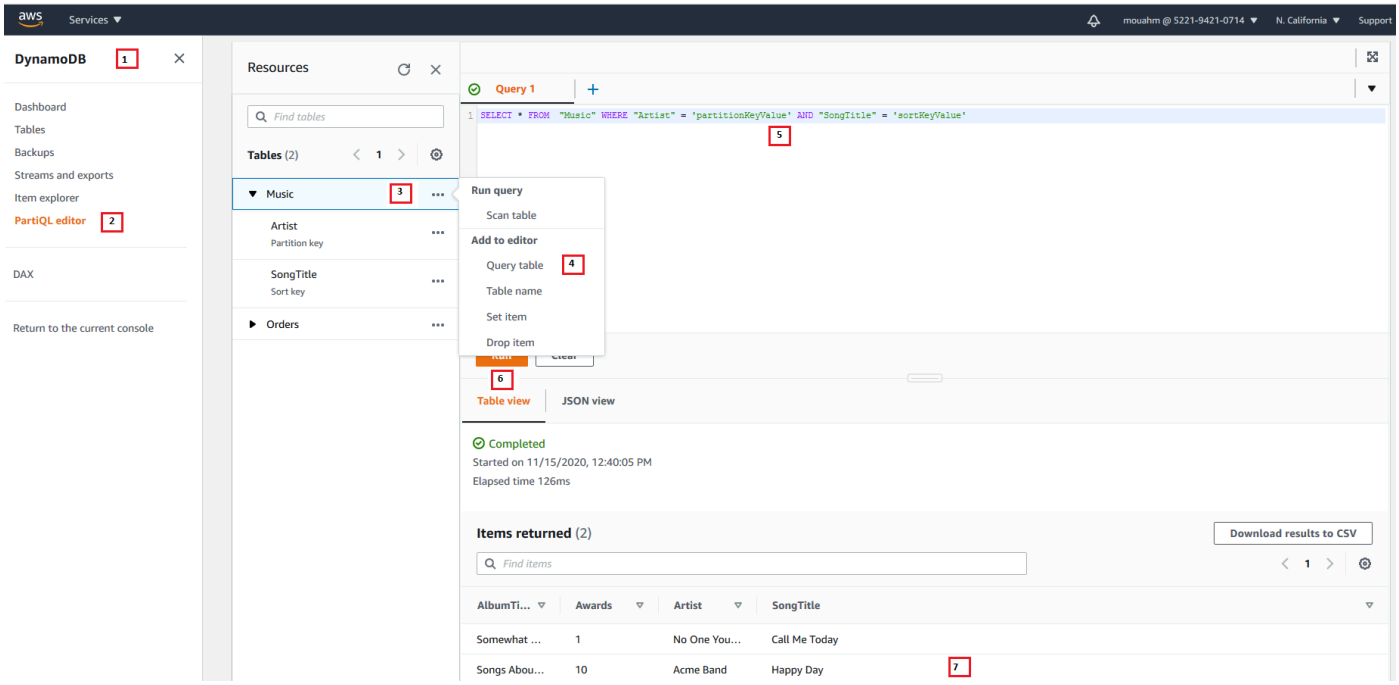
Cette section décrit comment utiliser PartiQL pour DynamoDB à partir de la console Amazon DynamoDB, de l'AWS Command Line Interface (AWS CLI) et des API DynamoDB.

Dans les exemples suivants, la table DynamoDB définie dans le didacticiel Mise en route avec DynamoDB est un prérequis.

Pour plus d'informations sur l'utilisation de la console DynamoDB, de l'AWS Command Line Interface ou des API DynamoDB pour accéder à DynamoDB, consultez Accès à DynamoDB.

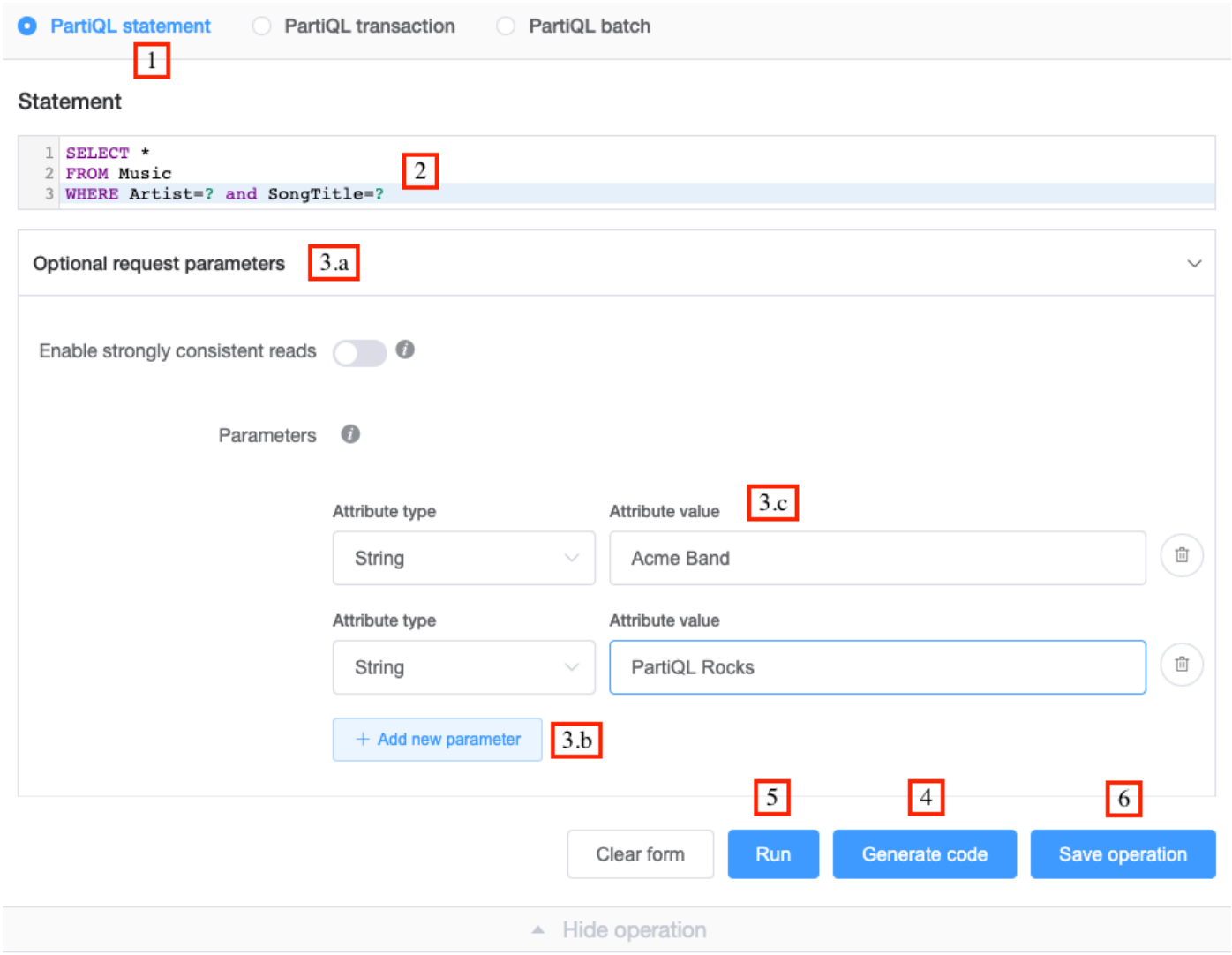
Pour télécharger et utiliser le NoSQL Workbench afin de créer des instructions PartiQL pour DynamoDB, choisissez PartiQL operations (Opérations PartiQL) dans l'angle supérieur droit du NoSQL Workbench pour DynamoDB Operation Builder (Créateur d'opérations).

Console



1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le panneau de navigation sur le côté gauche de la console, choisissez PartiQL editor (Editeur PartiQL).
3. Choisissez la table Music.
4. Choisissez Query table (Table de requête). Cette action génère une requête qui n'entraîne pas d'analyse de table complète.
5. Remplacez `partitionKeyValue` par la valeur de chaîne `Acme Band`. Remplacez `sortKeyValue` par la valeur de chaîne `Happy Day`.
6. Choisissez le bouton Run (Exécuter).
7. Vous pouvez afficher les résultats de la requête en choisissant les boutons Table view (Vue tableau) ou JSON view (Vue JSON).

NoSQL workbench



1. Choisissez PartiQL statement (Instruction PartiQL).
2. Entrez l'instruction PartiQL SELECT suivante

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. Pour spécifier une valeur pour les paramètres `Artist` et `SongTitle` :
   a. Choisissez Optional request parameters (Paramètres de demande facultatifs).
   b. Choisissez Add new parameters (Ajouter de nouveaux paramètres).
   c. Choisissez le type d'attribut string (chaîne) et la valeur `Acme Band`.

d. Répétez les étapes b et c, puis choisissez le type string (chaîne) et la valeur `PartiQL Rocks`.

4. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Sélectionnez votre langage souhaité dans les onglets affichés. Vous pouvez désormais copier ce code et l'utiliser dans votre application.

5. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).

AWS CLI

1. Créez un élément dans la table `Music` à l'aide de l'instruction INSERT PartiQL.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Extrayez un élément de la table Music à l'aide de l'instruction SELECT PartiQL.

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. Créez un élément dans la table `Music` à l'aide de l'instruction UPDATE PartiQL.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Ajoutez une valeur de liste pour un élément dans la table `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Supprimez une valeur de liste pour un élément dans la table `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Ajoutez un nouveau membre de mappage pour un élément dans la table `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Ajoutez un nouvel attribut d'ensemble de chaînes pour un élément dans la table `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Mettez à jour un attribut d'ensemble de chaînes pour un élément dans la table `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Supprimez un élément de la table `Music` à l'aide de l'instruction PartiQL DELETE.

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;
```

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
 SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
 SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
```

```
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
SongTitle=?", parameters));

            //Remove a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static List<AttributeValue> getPartiQLParameters() {
        List<AttributeValue> parameters = new ArrayList<AttributeValue>();
        parameters.add(new AttributeValue("Acme Band"));
        parameters.add(new AttributeValue("PartiQL Rocks"));
        return parameters;
```

```
   }

   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                       "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                       "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                       "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }
```

```
    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                                        "retrying. Error: " +
 rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                                        "Otherwise consider reducing frequency of
 requests or increasing provisioned capacity for your table or secondary index.
 Error: " +
                                        ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                                        "service, but for some reason, the service
 was not able to process it, and returned an error response instead. Investigate and
 " +
                                        "configure retry strategy. Error type: " +
 ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                                        "service, or the client was unable to parse
 the response from the service. Investigate and configure retry strategy. "+
                                        "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Types de données PartiQL pour DynamoDB

Le tableau suivant répertorie les types de données que vous pouvez utiliser avec PartiQL pour DynamoDB.

Type de données DynamoDB	Représentation PartiQL	Remarques
`Boolean`	TRUE | FALSE	Ne respecte pas la casse.
`Binary`	N/A	Uniquement pris en charge via un code.
`List`	[value1, value2,...]	Il n'existe aucune restriction quant aux types de données qui peuvent être stockés dans un type List (liste), et les éléments d'une liste ne doivent pas être du même type.
`Map`	{ 'name' : value }	Il n'existe aucune restriction quant aux types de données qui peuvent être stockés dans un type Map (mappage), et les éléments d'un mappage ne doivent pas être du même type.
`Null`	NULL	Ne respecte pas la casse.
`Number`	1, 1.0, 1e0	Les nombres peuvent être positifs, négatifs ou nuls. La précision maximum des nombres est de 38 chiffres.
`Number Set`	<<number1, number2>>	Les éléments d'un jeu de nombres doivent être de type Number (nombre).

Type de données DynamoDB	Représentation PartiQL	Remarques
`String Set`	<<’string1’, ’string2’>>	Les éléments d’un jeu de chaînes doivent être de type String (chaîne).
`String`	’string value’	Des apostrophes doivent être utilisées pour spécifier les valeurs de chaîne.

Exemples

L’instruction suivante montre comment insérer les types de données suivants : `String`, `Number`, `Map`, `List`, `Number Set` et `String Set`.

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

L’instruction suivante montre comment insérer de nouveaux éléments dans les types `Map`, `List`, `Number Set` et `String Set`, et modifier la valeur d’un type `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

L’instruction suivante montre comment supprimer des éléments des types `Map`, `List`, `Number Set` et `String Set`, et modifier la valeur d’un type `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType - 1
REMOVE ListType[1]
```

```
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

Pour plus d'informations, consultez Types de données DynamoDB.

Instructions PartiQL pour DynamoDB

Amazon DynamoDB prend en charge les instructions PartiQL suivantes.

> **Note**
>
> DynamoDB ne prend pas en charge toutes les instructions PartiQL.
> Cette référence fournit des exemples de syntaxe de base et d'utilisation des instructions partiQL que vous exécutez manuellement à l'aide de la AWS CLI commande or. APIs

Le langage de manipulation des données (Data manipulation language, DML) est l'ensemble d'instructions PartiQL que vous utilisez pour gérer les données dans des tables DynamoDB. Vous utilisez des instructions DML pour ajouter, modifier ou supprimer des données dans une table.

Les instructions DML et de langage de requête prises en charge sont les suivantes :

- Instructions PartiQL de sélection pour DynamoDB
- Instructions de mise à jour de PartiQL pour DynamoDB
- Instructions d'insertion de PartiQL pour DynamoDB
- Instructions de suppression de PartiQL pour DynamoDB

Les instructions Exécution de transactions avec PartiQL pour DynamoDB et Exécution d'opérations par lot avec PartiQL pour DynamoDB sont également prises en charge par PartiQL pour DynamoDB.

Instructions PartiQL de sélection pour DynamoDB

Utilisez l'instruction `SELECT` pour extraire des données d'une table dans Amazon DynamoDB.

L'utilisation de l'instruction `SELECT` peut entraîner une analyse de table complète si une condition IN avec une clé de partition n'est pas fournie dans la clause WHERE. Une opération d'analyse examine chaque élément en lien avec les valeurs demandées, et peut utiliser tout le débit approvisionné pour une table ou un index volumineux en une seule opération.

Si vous voulez éviter une analyse de table complète dans PartiQL, vous pouvez :

- Créer vos instructions SELECT de façon à ce qu'elle n'entraînent pas d'analyse de table complète en vous assurant que votre condition de clause WHERE est configurée en conséquence.
- Désactiver l'analyse de table complète à l'aide de la politique IAM spécifiée dans Exemple : autoriser les instructions de sélection et rejeter les instructions d'analyse de table complète dans PartiQL pour DynamoDB, dans le Manuel du développeur DynamoDB.

Pour plus d'informations, consultez Bonnes pratiques pour l'interrogation et l'analyse des données dans le Guide du développeur DynamoDB.

Rubriques

- Syntaxe
- Parameters
- Exemples

Syntaxe

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameters

expression

(Obligatoire) Projection formée à partir du caractère générique * ou d'une liste de projection d'un ou de plusieurs noms d'attributs ou chemins d'accès de documents de l'ensemble de résultats. Une expression peut être constituée d'appels à Utilisation de fonctions PartiQL avec DynamoDB ou de champs modifiés par Opérateurs arithmétiques, de comparaison et logiques PartiQL pour DynamoDB .

table

(Obligatoire) Nom de la table à interroger.

index

(Facultatif) Nom de l'index à interroger.

> **Note**
>
> Vous devez ajouter des guillemets doubles au nom de la table et au nom de l'index lorsque vous interrogez un index.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(Facultatif) Critères de sélection pour la requête.

> **Important**
>
> Pour s'assurer qu'une instruction `SELECT` n'entraîne pas une analyse de table complète, la condition de clause `WHERE` doit spécifier une clé de partition. Utilisez l'opérateur d'égalité ou IN (DANS).
> Par exemple, si vous avez une table `Orders` avec une clé de partition `OrderID` et des attributs autres que de clé, dont une `Address`, les instructions suivantes n'entraînent pas d'analyse de table complète :
>
> ```
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 and Address='some address'
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 or OrderID = 200
>
> SELECT *
> FROM "Orders"
> WHERE OrderID IN [100, 300, 234]
> ```
>
> En revanche, les instructions `SELECT` suivantes entraînent une analyse de table complète :

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

key

(Facultatif) Clé de hachage ou clé de tri à utiliser pour ordonner les résultats renvoyés. L'ordre par défaut est croissant (ASC). Spécifiez DESC si vous voulez que les résultats soient réordonnés dans l'ordre décroissant.

> **Note**
>
> Si vous omettez la clause WHERE, tous les éléments de la table sont extraits.

Exemples

La requête suivante renvoie un élément existant de la table `Orders` en spécifiant la clé de partition, `OrderID`, et en utilisant l'opérateur d'égalité.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

La requête suivante renvoie tous les éléments de la table `Orders` ayant une clé de partition spécifique, `OrderID`, et leurs valeurs en utilisant l'opérateur OR (OU).

```
SELECT OrderID, Total
FROM "Orders"
```

```
WHERE OrderID = 1 OR OrderID = 2
```

La requête suivante renvoie tous les éléments de la table `Orders` ayant une clé de partition spécifique, `OrderID`, et leurs valeurs en utilisant l'opérateur IN (DANS). Les résultats renvoyés sont dans l'ordre décroissant de la valeur de leur attribut de clé `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

La requête suivante affiche une analyse de table complète qui renvoie tous les éléments de la table `Orders` dont la valeur `Total` est supérieure à 500, où `Total` est un attribut autre que de clé.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

La requête suivante affiche une analyse de table complète qui renvoie tous les éléments de la table `Orders`dans une plage `Total` spécifique, en utilisant l'opérateur IN (DANS) et un attribut autre que de clé `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

La requête suivante affiche une analyse de table complète qui renvoie tous les éléments de la table `Orders`dans une plage `Total` spécifique, en utilisant l'opérateur BETWEEN (ENTRE) et un attribut autre que de clé `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

La requête suivante renvoie la première date à laquelle un Firestick a été utilisé, en spécifiant la clé de partition `CustomerID` et une clé de tri `MovieID` dans la condition de clause WHERE, et en utilisant des chemins d'accès de document dans la clause SELECT.

```
SELECT Devices.FireStick.DateWatched[0]
```

```
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

La requête suivante affiche une analyse de table complète qui renvoie la liste des éléments pour lesquels un Firestick a été utilisé pour la première fois après le 24/12/19, en utilisant des chemins de document dans la condition de la clause WHERE.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Instructions de mise à jour de PartiQL pour DynamoDB

Utilisez l'instruction UPDATE pour modifier la valeur d'un ou de plusieurs attributs dans un élément d'une table Amazon DynamoDB.

> **Note**
>
> Vous ne pouvez mettre à jour qu'un seul élément à la fois et ne pouvez pas émettre une instruction PartiQL pour DynamoDB qui met à jour plusieurs éléments. Pour plus d'informations sur la mise à jour de plusieurs éléments, consultez Exécution de transactions avec PartiQL pour DynamoDB ou Exécution d'opérations par lot avec PartiQL pour DynamoDB.

Rubriques

- Syntaxe
- Parameters
- Valeur renvoyée
- Exemples

Syntaxe

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameters

table

(Obligatoire) Table contenant les données à modifier.

path

(Obligatoire) Nom d'attribut ou chemin d'accès de document à créer ou à modifier.

data

(Obligatoire) Valeur d'attribut ou résultat d'une opération.

Opérations prises en charge à utiliser avec SET :

- LIST_APPEND : ajoute une valeur à un type de liste.
- SET_ADD : ajoute une valeur à un ensemble de nombres ou de chaînes.
- SET_DELETE : supprime une valeur d'un ensemble de nombres ou de chaînes.

condition

(Obligatoire) Critères de sélection de l'élément à modifier. Le résultat de cette condition doit être une seule valeur de clé primaire.

returnvalues

(Facultatif) Utilisez `returnvalues` si vous souhaitez obtenir les attributs de l'élément avant ou après sa mise à jour. Les valeurs valides sont :

- `ALL OLD *` – Renvoie tous les attributs de l'élément avant l'opération de mise à jour.
- `MODIFIED OLD *` – Renvoie uniquement les attributs mis à jour avant l'opération de mise à jour.
- `ALL NEW *` – Renvoie tous les attributs de l'élément après l'opération de mise à jour.
- `MODIFIED NEW *` – Renvoie uniquement les attributs mis à jour après l'opération `UpdateItem`.

Valeur renvoyée

Cette instruction ne renvoie de valeur que si le paramètre `returnvalues` est spécifié.

> **Note**
>
> Si la clause WHERE de l'instruction UPDATE ne produit le résultat true pour aucun élément de la table DynamoDB, `ConditionalCheckFailedException` est renvoyé.

Exemples

Mettez à jour une valeur d'attribut d'un élément existant. Si l'attribut n'existe pas, il est créé.

La requête suivante met à jour un élément dans la table `"Music"` en ajoutant un attribut de type Number (`AwardsWon`) et un attribut de type Map (`AwardDetail`).

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Vous pouvez ajouter `RETURNING ALL OLD *` pour renvoyer les attributs tels qu'ils apparaissaient avant l'opération `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Ceci renvoie les informations suivantes :

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Vous pouvez ajouter `RETURNING ALL NEW *` pour renvoyer les attributs tels qu'ils sont apparus après l'opération `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Ceci renvoie les informations suivantes :

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

La requête suivante met à jour un élément dans la table `"Music"` en effectuant un ajout à une liste `AwardDetail.Grammys`.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La requête suivante met à jour un élément dans la table `"Music"` en effectuant une suppression d'une liste `AwardDetail.Grammys`.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La requête suivante met à jour un élément dans la table `"Music"` en ajoutant `BillBoard` au mappage `AwardDetail`.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La requête suivante met à jour un élément dans la table `"Music"` en ajoutant l'attribut d'ensemble de chaînes `BandMembers`.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La requête suivante met à jour un élément dans la table `"Music"` en ajoutant `newbandmember` à l'ensemble de chaînes `BandMembers`.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Instructions de suppression de PartiQL pour DynamoDB

Utilisez l'instruction `DELETE` pour supprimer un élément existant de votre table Amazon DynamoDB.

> **Note**
>
> Vous ne pouvez supprimer qu'un seul élément à la fois. Vous ne pouvez pas émettre une seule instruction PartiQL pour DynamoDB qui supprime plusieurs éléments. Pour plus d'informations sur la suppression de plusieurs éléments, consultez Exécution de transactions avec PartiQL pour DynamoDB ou Exécution d'opérations par lot avec PartiQL pour DynamoDB.

Rubriques

- Syntaxe
- Parameters
- Valeur renvoyée
- Exemples

Syntaxe

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameters

table

(Obligatoire) Table DynamoDB contenant l'élément à supprimer.

condition

(Obligatoire) Critères de sélection de l'élément à supprimer. Le résultat de cette condition doit être une seule valeur de clé primaire.

returnvalues

(Facultatif) Utilisez `returnvalues` si vous souhaitez obtenir les attributs de l'élément avant sa suppression. Les valeurs valides sont :

- `ALL OLD *` – Le contenu de l'ancien élément est renvoyé.

Valeur renvoyée

Cette instruction ne renvoie de valeur que si le paramètre `returnvalues` est spécifié.

> **Note**
>
> Si la table DynamoDB ne contient aucun élément dont la même clé primaire est la même que celle de l'élément pour lequel l'instruction DELETE est émise, le résultat SUCCESS est renvoyé avec 0 élément supprimé. Si la table contient un élément avec la même clé primaire,

mais que le résultat de la condition dans la clause WHERE de l'instruction DELETE est false (faux), l'erreur `ConditionalCheckFailedException` est renvoyée.

Exemples

La requête suivante interroge un élément dans la table `"Music"`.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Vous pouvez ajouter le paramètre `RETURNING ALL OLD *` pour renvoyer les données supprimées.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

L'instruction `Delete` renvoie désormais ce qui suit :

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Instructions d'insertion de PartiQL pour DynamoDB

Utilisez l'instruction `INSERT` pour ajouter un élément à une table dans Amazon DynamoDB.

> **Note**
>
> Vous ne pouvez insérer qu'un seul élément à la fois et ne pouvez pas émettre une instruction PartiQL pour DynamoDB qui insère plusieurs éléments. Pour plus d'informations sur l'insertion de plusieurs éléments, consultez Exécution de transactions avec PartiQL pour DynamoDB ou Exécution d'opérations par lot avec PartiQL pour DynamoDB.

Rubriques

- Syntaxe
- Parameters
- Valeur renvoyée
- Exemples

Syntaxe

Insérez un seul élément.

```
INSERT INTO table VALUE item
```

Parameters

table

(Obligatoire) Table dans laquelle vous souhaitez insérer les données. La table doit déjà exister.

item

(Obligatoire) Elément DynamoDB valide représenté sous la forme d'un Tuple PartiqL. Vous devez spécifier un seul élément. Chaque nom d'attribut dans l'élément est sensible à la casse et peut être indiqué par des apostrophes (`'...'`) dans PartiQL.

Les valeurs de chaîne sont également indiquées par des apostrophes (`'...'`) dans PartiQL.

Valeur renvoyée

Cette instruction ne renvoie aucune valeur.

> **Note**
>
> Si la table DynamoDB contient déjà un élément avec la même clé primaire que la clé primaire de l'élément inséré, l'erreur `DuplicateItemException` est renvoyée.

Exemples

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Utilisation de fonctions PartiQL avec DynamoDB

PartiQL dans Amazon DynamoDB prend en charge les variantes intégrées suivantes des fonctions standard SQL.

> **Note**
>
> Les fonctions SQL qui ne figurent pas dans cette liste ne sont actuellement pas prises en charge dans DynamoDB.

Fonctions d'agrégation

- Utilisation de la fonction SIZE avec PartiQL pour Amazon DynamoDB

Fonctions conditionnelles

- Utilisation de la fonction EXISTS avec PartiQL pour DynamoDB
- Utilisation de la fonction ATTRIBUTE_TYPE avec PartiQL pour DynamoDB
- Utilisation de la fonction BEGINS_WITH avec PartiQL pour DynamoDB
- Utilisation de la fonction CONTAINS avec PartiQL pour DynamoDB
- Utilisation de la fonction MISSING avec PartiQL pour DynamoDB

Utilisation de la fonction EXISTS avec PartiQL pour DynamoDB

Vous pouvez utiliser la fonction EXISTS pour effectuer la même opération que la fonction `ConditionCheck` dans l'API TransactWriteItems. Vous ne pouvez utiliser la fonction EXISTS que dans les transactions.

Pour une valeur donnée, la fonction renvoie `TRUE` si la valeur est une collection non vide. Sinon, la valeur renvoyée est `FALSE`.

> **Note**
>
> Vous ne pouvez utiliser cette fonction que dans les opérations transactionnelles.

Syntaxe

```
EXISTS ( statement )
```

Arguments

statement

(Obligatoire) Instruction SELECT que la fonction évalue.

> **Note**
>
> L'instruction SELECT doit spécifier une clé primaire complète et une autre condition.

Type de retour

`bool`

Exemples

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Utilisation de la fonction BEGINS_WITH avec PartiQL pour DynamoDB

La fonction renvoie TRUE si l'attribut spécifié commence par une sous-chaîne particulière.

Syntaxe

```
begins_with(path, value )
```

Arguments

path

(Obligatoire) Chemin d'accès du nom d'attribut ou du document à utiliser.

valeur

(Obligatoire) Chaîne à rechercher.

Type de retour

`bool`

Exemples

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Utilisation de la fonction MISSING avec PartiQL pour DynamoDB

La fonction renvoie TRUE si l'élément ne contient pas l'attribut spécifié. Seuls des opérateurs d'égalité et d'inégalité peuvent être utilisés avec cette fonction.

Syntaxe

```
 attributename IS | IS NOT  MISSING
```

Arguments

attributename

(Obligatoire) Nom d'attribut à rechercher.

Type de retour

`bool`

Exemples

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Utilisation de la fonction ATTRIBUTE_TYPE avec PartiQL pour DynamoDB

La fonction renvoie TRUE si l'attribut dans le chemin d'accès spécifié est d'un type de données particulier.

Syntaxe

```
attribute_type( attributename, type )
```

Arguments

attributename

(Obligatoire) Nom d'attribut à utiliser.

type

(Obligatoire) Type d'attribut à vérifier. Pour obtenir la liste des valeurs valides, consultez attribute_type DynamoDB.

Type de retour

`bool`

Exemples

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Utilisation de la fonction CONTAINS avec PartiQL pour DynamoDB

La fonction renvoie TRUE si l'attribut spécifié par le chemin d'accès est l'un des attributs suivants :

- Une chaîne contenant une sous-chaîne particulière.
- Un ensemble contenant un élément particulier.

Pour plus d'informations, consultez la fonction DynamoDB contains.

Syntaxe

```
contains( path, substring )
```

Arguments

path

(Obligatoire) Chemin d'accès du nom d'attribut ou du document à utiliser.

substring

(Obligatoire) Sous-chaîne d'attribut ou membre d'ensemble à vérifier. Pour plus d'informations, consultez la fonction DynamoDB contains.

Type de retour

`bool`

Exemples

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Utilisation de la fonction SIZE avec PartiQL pour Amazon DynamoDB

Renvoie un nombre représentant la taille en octets d'un attribut. Les types de données valides à utiliser avec la fonction Size sont les suivants. Pour plus d'informations, consultez la fonction DynamoDB size.

Syntaxe

```
size( path)
```

Arguments

`path`

(Obligatoire) Chemin d'accès du nom d'attribut ou du document à utiliser.

Pour les types pris en charge, consultez la fonction DynamoDB size.

Type de retour

`int`

Exemples

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

Opérateurs arithmétiques, de comparaison et logiques PartiQL pour DynamoDB

PartiQL dans Amazon DynamoDB prend en charge les opérateurs standard SQL suivants.

> **Note**
>
> Les opérateurs SQL qui ne figurent pas dans cette liste ne sont actuellement pas pris en charge dans DynamoDB.

Opérateurs arithmétiques

Opérateur	Description
+	Addition
-	Soustraction

Opérateurs de comparaison

Opérateur	Description
=	Egal à
<>	Non égal à
!=	Non égal à
>	Supérieure à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à

Opérateurs logiques

Opérateur	Description
`AND`	`TRUE` si toutes les conditions séparées par `AND` sont `TRUE`
`BETWEEN`	`TRUE` si l'opérande est comprise dans la plage des comparaisons. Cet opérateur inclut les limites inférieure et supérieure des opérandes auxquels vous l'appliquez.
`IN`	`TRUE` si l'opérande est égale à l'une d'une liste d'expressions (à 50 valeurs d'attribut de hachage maximum ou à 100 valeurs d'attribut non clé maximum)
`IS`	`TRUE` si l'opérande est un type de données PartiQL spécifique, dont `NULL` ou `MISSING`
`NOT`	Inverse la valeur d'une expression booléenne donnée
`OR`	`TRUE` si une ou plusieurs des conditions séparées par `OR` sont `TRUE`

Pour plus d'informations sur l'utilisation des opérateurs logiques, consultez Comparaisons et Évaluations logiques.

Exécution de transactions avec PartiQL pour DynamoDB

Cette section décrit comment utiliser des transactions avec PartiQL pour DynamoDB. Les transactions PartiQL sont limitées à 100 instructions (actions) au total.

Pour plus d'informations sur les transactions DynamoDB, consultez Gestion des flux de travail complexes avec les transactions DynamoDB.

> **Note**
>
> La transaction entière doit être composée d'instructions de lecture ou d'instructions d'écriture. Vous ne pouvez pas mélanger les deux dans une seule transaction. La fonction EXISTS est une exception. Vous pouvez l'utiliser pour vérifier l'état d'attributs spécifiques de l'article de la même manière que `ConditionCheck` dans le cadre de l'opération TransactWriteItemsAPI.

Rubriques

- Syntaxe
- Parameters
- Valeurs renvoyées
- Exemples

Syntaxe

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Obligatoire) Instruction prise en charge PartiQL pour DynamoDB.

> **Note**
>
> La transaction entière doit être composée d'instructions de lecture ou d'instructions d'écriture. Vous ne pouvez pas mélanger les deux dans une seule transaction.

parametertype

(Facultatif) Type DynamoDB, si des paramètres ont été utilisés lors de la spécification de l'instruction PartiQL.

parametervalue

(Facultatif) Valeur de paramètre si des paramètres ont été utilisés lors de la spécification de l'instruction PartiQL.

Valeurs renvoyées

Cette instruction ne renvoie aucune valeur pour les opérations d'écriture (INSERT, UPDATE ou DELETE). Toutefois, elle renvoie différentes valeurs pour les opérations de lecture (SELECT) en fonction des conditions spécifiées dans la clause WHERE.

> **Note**
>
> Si l'une des opérations singleton INSERT, UPDATE ou DELETE renvoie une erreur, les transactions sont annulées avec l'exception `TransactionCanceledException`, et le code de motif d'annulation inclut les erreurs des opérations singleton individuelles.

Exemples

L'exemple suivant exécute plusieurs instructions sous forme de transaction.

AWS CLI

1. Enregistrez le code JSON suivant dans un fichier nommé partiql.json.

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
```

```
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. Dans une invite de commande, exécutez la commande suivante.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create ExecuteTransactionRequest
            ExecuteTransactionRequest executeTransactionRequest =
 createExecuteTransactionRequest();
            ExecuteTransactionResult executeTransactionResult =
 dynamoDB.executeTransaction(executeTransactionRequest);
            System.out.println("ExecuteTransaction successful.");
            // Handle executeTransactionResult

        } catch (Exception e) {
            handleExecuteTransactionErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static ExecuteTransactionRequest createExecuteTransactionRequest() {
        ExecuteTransactionRequest request = new ExecuteTransactionRequest();

        // Create statements
        List<ParameterizedStatement> statements = getPartiQLTransactionStatements();
```

```
        request.setTransactStatements(statements);
        return request;
    }

    private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
        List<ParameterizedStatement> statements = new
ArrayList<ParameterizedStatement>();

        statements.add(new ParameterizedStatement()
                                .withStatement("EXISTS(SELECT * FROM "Music" where
Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));

        statements.add(new ParameterizedStatement()
                                .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                                .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));

        statements.add(new ParameterizedStatement()
                                .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

        return statements;
    }

    // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteTransactionErrors(Exception exception) {
        try {
            throw exception;
        } catch (TransactionCanceledException tce) {
            System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
        } catch (TransactionInProgressException tipe) {
            System.out.println("The transaction with the given request token is
already in progress, consider changing " +
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
```

```
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }

   private static void handleCommonErrors(Exception exception) {
       try {
           throw exception;
       } catch (InternalServerErrorException isee) {
           System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
       } catch (RequestLimitExceededException rlee) {
           System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
       } catch (ProvisionedThroughputExceededException ptee) {
           System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
       } catch (ResourceNotFoundException rnfe) {
           System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
       } catch (AmazonServiceException ase) {
           System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
Error message: " + ase.getErrorMessage());
       } catch (AmazonClientException ace) {
           System.out.println("An AmazonClientException occurred, indicates that
the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
       } catch (Exception e) {
           System.out.println("An exception occurred, investigate and configure
retry strategy. Error: " + e.getMessage());
       }
   }
```

```
}
```

L'exemple suivant montre les différentes valeurs renvoyées lorsque DynamoDB lit des éléments avec des conditions différentes spécifiées dans la clause WHERE.

AWS CLI

1. Enregistrez le code JSON suivant dans un fichier nommé partiql.json.

```
[
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. commande suivante, dans une invite de commande.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. La réponse suivante est renvoyée :

```
{
    "Responses": [
        // Item exists and projected attribute exists
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
```

```
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
}
```

Exécution d'opérations par lot avec PartiQL pour DynamoDB

Cette section décrit comment utiliser des opérations par lot avec PartiQL pour DynamoDB.

> **Note**
>
> - Le lot entier doit être composé d'instructions de lecture ou d'instructions d'écriture. Vous ne pouvez pas mélanger les deux dans une seule opération par lot.
> - `BatchExecuteStatement` et `BatchWriteItem` ne peuvent pas exécuter plus de 25 instructions par lot.
> - `BatchExecuteStatement` utilise `BatchGetItem` ce qui prend une liste de clés primaires dans des instructions séparées.

Rubriques

- Syntaxe
- Parameters
- Exemples

Syntaxe

```
[
  {
```

```
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Obligatoire) Instruction prise en charge PartiQL pour DynamoDB.

> **Note**
>
> - Le lot entier doit être composé d'instructions de lecture ou d'instructions d'écriture. Vous ne pouvez pas mélanger les deux dans une seule opération par lot.
> - `BatchExecuteStatement` et `BatchWriteItem` ne peuvent pas exécuter plus de 25 instructions par lot.

parametertype

(Facultatif) Type DynamoDB, si des paramètres ont été utilisés lors de la spécification de l'instruction PartiQL.

parametervalue

(Facultatif) Valeur de paramètre si des paramètres ont été utilisés lors de la spécification de l'instruction PartiQL.

Exemples

AWS CLI

1. Enregistrez le JSON suivant dans un fichier nommé partiql.json

```
[
   {
  "Statement": "INSERT INTO Music VALUE {'Artist':?,'SongTitle':?}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
  "Statement": "UPDATE Music SET AwardsWon=1, AwardDetail={'Grammys':[2020,
 2018]} WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
    }
]
```

2. Dans une invite de commande, exécutez la commande suivante.

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }
```

```
   private static AmazonDynamoDB createDynamoDbClient(String region) {

       return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
   }

   private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
{
       BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

       // Create statements
       List<BatchStatementRequest> statements = getPartiQLBatchStatements();

       request.setStatements(statements);
       return request;
   }

   private static List<BatchStatementRequest> getPartiQLBatchStatements() {
       List<BatchStatementRequest> statements = new
ArrayList<BatchStatementRequest>();

       statements.add(new BatchStatementRequest()
                              .withStatement("INSERT INTO Music value
{'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

       statements.add(new BatchStatementRequest()
                              .withStatement("UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL
Rocks'"));

       return statements;
   }

   // Handles errors during BatchExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleBatchExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (Exception e) {
           // There are no API specific errors to handle for BatchExecuteStatement,
common DynamoDB API errors are handled below
           handleCommonErrors(e);
       }
```

```
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Politiques de sécurité IAM avec PartiQL pour DynamoDB

Les autorisations suivantes sont requises :

- Pour lire des éléments à l'aide de PartiQL pour DynamoDB, vous devez disposer de l'autorisation `dynamodb:PartiQLSelect` sur la table ou l'index.
- Pour insérer des éléments à l'aide de PartiQL pour DynamoDB, vous devez disposer de l'autorisation `dynamodb:PartiQLInsert` sur la table ou l'index.
- Pour mettre à jour des éléments à l'aide de PartiQL pour DynamoDB, vous devez disposer de l'autorisation `dynamodb:PartiQLUpdate` sur la table ou l'index.
- Pour supprimer des éléments à l'aide de PartiQL pour DynamoDB, vous devez disposer de l'autorisation `dynamodb:PartiQLDelete` sur la table ou l'index.

Exemple : autoriser toutes les instructions PartiQL pour DynamoDB () sur une table Select/Insert/Update/Delete

La politique IAM suivante accorde les autorisations nécessaires pour exécuter toutes les instructions PartiQL pour DynamoDB sur une table.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Exemple : Autoriser les instructions PartiQL select pour DynamoDB sur une table

La politique IAM suivante accorde les autorisations nécessaires pour exécuter l'instruction `select` sur une table spécifique.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Exemple : Autoriser les instructions PartiQL insert pour DynamoDB sur un index

La politique IAM suivante accorde les autorisations nécessaires pour exécuter l'instruction `insert` sur un index spécifique.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
```

```
      }
   ]
}
```

Exemple : Autoriser les instructions transactionnelles PartiQL pour DynamoDB sur une table

La politique IAM suivante accorde les autorisations nécessaires pour exécuter uniquement des instructions transactionnelles sur une table spécifique.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

Exemple : Autoriser les lectures et d'écritures non transactionnelles PartiQL pour DynamoDB, et bloquer les instructions de lectures et d'écritures transactionnelles PartiQL sur une table.

La politique IAM suivante accorde des autorisations pour exécuter des lectures et des écritures non transactionnelles PartiQL pour DynamoDB tout en bloquant les lectures et écritures transactionnelles PartiQL pour DynamoDB.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
```

```
      }
   ]
}
```

Exemple : autoriser les instructions de sélection et rejeter les instructions d'analyse de table complète dans PartiQL pour DynamoDB

La politique IAM suivante accorde des autorisations pour exécuter l'instruction `select` sur une table spécifique tout en bloquant les instructions `select` qui entraînent une analyse de table complète.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
         "Condition":{
            "Bool":{
               "dynamodb:FullTableScan":[
                  "true"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ]
      }
   ]
```

```
}
```

Utilisation d'éléments : Java

L'API de document AWS SDK pour Java vous permet d'effectuer des opérations CRUD (création, lecture, mise à jour et suppression) classiques sur des éléments Amazon DynamoDB dans une table.

> **Note**
>
> Le kit SDK pour Java fournit également un modèle de persistance des objets qui vous permet de mapper vos classes côté client à des tables DynamoDB. Cette approche peut réduire la quantité de code que vous devez écrire. Pour en savoir plus, consultez Java 1.x : DynamoDBMapper.

Cette section comporte les exemples Java pour exécuter plusieurs actions d'élément d'API de document Java et plusieurs exemples opérationnels.

Rubriques

- Insertion d'un élément
- Obtention d'un élément
- Écriture par lots : insertion et suppression de plusieurs éléments
- Obtention par lots : obtention de plusieurs éléments
- Mise à jour d'un élément
- Suppression d'un élément
- Exemple : opérations CRUD à l'aide de l'API de document AWS SDK pour Java
- Exemple : opérations par lots à l'aide de l'API de document AWS SDK pour Java
- Exemple : gestion d'attributs de type binaire à l'aide de l'API de document de AWS SDK pour Java

Insertion d'un élément

La méthode `putItem` stocke un élément dans une table. Si l'élément existe, il remplace la totalité de l'élément. Au lieu de remplacer l'élément entier, vous pouvez mettre à jour uniquement des attributs spécifiques à l'aide de la méthode `updateItem`. Pour en savoir plus, consultez Mise à jour d'un élément.

Java v2

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval> <awards>
 <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is placed
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
```

```
                    SongTitleVal - The value of the song title (for example, Happy
Day).
                **Warning** This program will  place an item that you specify into a
table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {
```

```
        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The request
 id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Java v1

Procédez comme suit :

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table` pour représenter la table que vous souhaitez utiliser.
3. Création d'une instance de la classe `Item` pour représenter le nouvel élément. Vous devez spécifier la clé primaire du nouvel élément et ses attributs.
4. Appelez la méthode `putItem` de l'objet `Table` à l'aide de l'objet `Item` que vous avez créé à l'étape précédente.

L'exemple de code Java suivant illustre les tâches précédentes. Le code écrit un nouvel élément dans la table `ProductCatalog`.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();

List<String> fiveStarReviews = new ArrayList<String>();
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
```

```
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

Dans l'exemple précédent, l'élément comporte des attributs scalaires (`String`, `Number`, `Boolean` et `Null`), des ensembles (`String Set`) et des types de documents (`List`, `Map`).

Spécification de paramètres facultatifs

Outre les paramètres obligatoires, vous pouvez également spécifier des paramètres facultatifs de la méthode `putItem`. Par exemple, l'extrait de code Java suivant spécifie une condition pour le chargement de l'élément à l'aide d'un paramètre facultatif. Si la condition que vous spécifiez n'est pas remplie, AWS SDK pour Java lève une `ConditionalCheckFailedException`. L'extrait de code spécifie les paramètres facultatifs suivants dans la méthode `putItem` :

- Une `ConditionExpression` qui définit les conditions de la demande. Le code définit la condition selon laquelle l'élément existant ayant la même clé primaire est remplacé seulement s'il comporte un attribut ISBN égal à une valeur spécifique.
- Un mappage pour `ExpressionAttributeValues` utilisé dans la condition. Dans ce cas, une seule substitution est nécessaire : l'espace réservé `:val` dans l'expression de condition est remplacé lors de l'exécution par la valeur ISBN réelle à vérifier.

L'exemple suivant ajoute un nouvel élément livre à l'aide de ces paramètres facultatifs.

Example

```
Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));
```

```
Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem et documents JSON

Vous pouvez stocker un document JSON comme un attribut dans une table DynamoDB. Pour ce faire, utilisez la méthode `withJSON` de `Item`. Cette méthode analyse le document JSON et mappe chaque élément à un type de données DynamoDB natif.

Supposons que vous souhaitiez stocker le document JSON suivant, comportant des fournisseurs qui peuvent exécuter des commandes pour un produit spécifique.

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
}
```

La méthode `withJSON` vous permet de stocker cela dans la table `ProductCatalog`, dans un attribut `Map` nommé `VendorInfo`. L'extrait de code Java suivant montre comment procéder.

```
// Convert the document into a String.  Must escape all double-quotes.
```

```
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

Obtention d'un élément

Récupérez un élément unique à l'aide de la méthode `getItem` d'un objet `Table`. Procédez comme suit :

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table` pour représenter la table que vous souhaitez utiliser.
3. Appelez la méthode `getItem` de l'instance `Table`. Vous devez spécifier la clé primaire de l'élément que vous souhaitez récupérer.

L'exemple de code Java suivant illustre les tâches précédentes. Le code obtient l'élément qui possède la clé de partition spécifiée.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");
```

```
Item item = table.getItem("Id", 210);
```

Spécification de paramètres facultatifs

Outre les paramètres obligatoires, vous pouvez également spécifier des paramètres facultatifs de la méthode `getItem`. Par exemple, l'extrait de code Java suivant récupère uniquement une liste précise d'attributs et spécifie des lectures fortement cohérentes à l'aide d'une méthode facultative. (Pour en savoir plus sur la cohérence en lecture, consultez Cohérence en lecture DynamoDB.)

Une `ProjectionExpression` vous permet de récupérer uniquement certains attributs ou éléments, plutôt qu'un élément entier. Une `ProjectionExpression` peut spécifier des attributs de niveau supérieur ou imbriqués à l'aide de chemins d'accès au document. Pour en savoir plus, consultez Utilisation d'expressions de projection dans DynamoDB.

Les paramètres de la méthode `getItem` ne vous permettent pas de spécifier une cohérence en lecture. Cependant, vous pouvez créer une `GetItemSpec`, qui fournit un accès complet à l'ensemble des entrées de l'opération `GetItem` de bas niveau. L'exemple de code ci-dessous crée une `GetItemSpec` et utilise cette spécification comme entrée de la méthode `getItem`.

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);

System.out.println(item.toJSONPretty());
```

Imprimez un `Item` dans un format contrôlable de visu à l'aide de la méthode `toJSONPretty`. La sortie de l'exemple précédent se présente comme suit.

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
```

```
  "Id" : 123,
  "Title" : "20-Bicycle 123"
}
```

GetItem et documents JSON

Dans la section PutItem et documents JSON,vous stockez un document JSON dans un attribut `Map` nommé `VendorInfo`. La méthode `getItem` vous permet d'extraire la totalité du document au format JSON. Vous pouvez également extraire uniquement certains éléments du document à l'aide de la notation du chemin d'accès au document. L'extrait de code Java suivant illustre ces techniques.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);

System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

La sortie de l'exemple précédent se présente comme suit.

```
All vendor info:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

> **Note**
>
> La méthode `toJSON` vous permet de convertir tout élément (ou ses attributs) en une chaîne au format JSON. L'extrait de code suivant récupère plusieurs attributs de niveau supérieur et imbriqués et imprime les résultats au format JSON.
>
> ```
> GetItemSpec spec = new GetItemSpec()
>     .withPrimaryKey("Id", 210)
>     .withProjectionExpression("VendorInfo.V01, Title, Price");
>
> Item item = table.getItem(spec);
> System.out.println(item.toJSON());
> ```
>
> Le résultat se présente comme suit.
>
> ```
> {"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
> ["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
> ```

Écriture par lots : insertion et suppression de plusieurs éléments

L'écriture par lots désigne l'insertion et la suppression de plusieurs éléments dans un lot. La méthode `batchWriteItem` vous permet d'insérer et de supprimer plusieurs éléments d'une ou de plusieurs tables en un seul appel. Pour insérer ou supprimer plusieurs éléments à l'aide de l'API de document AWS SDK pour Java, procédez comme suit :

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `TableWriteItems` qui décrit toutes les opérations d'insertion et de suppression pour une table. Si vous souhaitez écrire sur plusieurs tables en une opération unique d'écriture par lots, vous devez créer une instance `TableWriteItems` par table.
3. Appelez la méthode `batchWriteItem` en fournissant les objets `TableWriteItems` que vous avez créés à l'étape précédente.
4. Traitez la réponse. Vous devez vérifier si des éléments de demandes non traités ont été renvoyés dans la réponse. Cela peut se produire si vous atteignez le quota de débit alloué ou en cas d'autre erreur temporaire. En outre, DynamoDB limite la taille de demande et le nombre d'opérations que vous pouvez spécifier dans une demande. Si vous dépassez ces

limites, DynamoDB rejette la demande. Pour en savoir plus, consultez Quotas dans Amazon DynamoDB.

L'exemple de code Java suivant illustre les tâches précédentes. L'exemple effectue une opération `batchWriteItem` sur deux tables : `Forum` et `Thread`. Les objets `TableWriteItems` correspondants permettent de définir les actions suivantes :

- Insérer un élément dans la table `Forum`
- Insérer et supprimer un élément de la table `Thread`

Le code appelle ensuite `batchWriteItem` pour effectuer l'opération.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

Pour obtenir un exemple pratique, consultez Exemple : opérations d'écriture par lots à l'aide de l'API de document AWS SDK pour Java.

Obtention par lots : obtention de plusieurs éléments

La méthode `batchGetItem` vous permet de récupérer plusieurs éléments d'une ou de plusieurs tables. Vous pouvez récupérer un seul élément à l'aide de la méthode `getItem`.

Procédez comme suit :

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `TableKeysAndAttributes` qui décrit une liste de valeurs de clé primaire à récupérer dans une table. Si vous souhaitez lire depuis plusieurs tables dans une opération unique d'obtention par lots, vous devez créer une instance `TableKeysAndAttributes` par table.
3. Appelez la méthode `batchGetItem` en fournissant les objets `TableKeysAndAttributes` que vous avez créés à l'étape précédente.

L'exemple de code Java suivant illustre les tâches précédentes. L'exemple récupère deux éléments dans la table `Forum` et trois éléments dans la table `Thread`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);

for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

Spécification de paramètres facultatifs

Outre les paramètres obligatoires, vous pouvez également spécifier des paramètres facultatifs lorsque vous utilisez `batchGetItem`. Par exemple, vous pouvez fournir une `ProjectionExpression` avec chaque `TableKeysAndAttributes` que vous définissez. Cela vous permet de spécifier les attributs que vous souhaitez récupérer à partir de la table.

L'exemple de code suivant extrait trois éléments de la table `Forum`. Le paramètre `withProjectionExpression` spécifie que seul l'attribut `Threads` doit être récupéré.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Mise à jour d'un élément

La méthode `updateItem` d'un objet `Table` peut mettre à jour les valeurs d'attribut existantes, ajouter de nouveaux attributs ou supprimer des attributs d'un élément existant.

La méthode `updateItem` se comporte comme suit :

- Si un élément n'existe pas (aucun élément dans la table avec la clé primaire spécifiée), `updateItem` ajoute un nouvel élément à la table
- Si un élément existe, `updateItem` effectue la mise à jour comme indiqué par le paramètre `UpdateExpression`.

> **Note**
>
> Il est également possible de « mettre à jour » un élément à l'aide de `putItem`. Par exemple, si vous appelez `putItem` pour ajouter un élément à la table, mais qu'il y a déjà un élément avec la clé primaire spécifiée, `putItem` remplace l'élément entier. S'il y a des attributs dans l'élément existant non spécifiés dans l'entrée, `putItem` supprime ces attributs de l'élément.

> En général, nous vous recommandons d'utiliser `updateItem` chaque fois que vous souhaitez modifier des attributs d'élément. La méthode `updateItem` modifie uniquement les attributs d'élément que vous spécifiez dans l'entrée et les autres attributs de l'élément demeurent inchangés.

Procédez comme suit :

1. Créez une instance de la classe `Table` pour représenter la table que vous souhaitez utiliser.
2. Appelez la méthode `updateTable` de l'instance `Table`. Vous devez spécifier la clé primaire de l'élément que vous souhaitez récupérer, avec une `UpdateExpression` qui décrit les attributs à modifier et comment les modifier.

L'exemple de code Java suivant illustre les tâches précédentes. Le code met à jour un élément livre de la table `ProductCatalog`. Il ajoute un nouvel auteur à l'ensemble `Authors` et supprime l'attribut `ISBN` existant. Il permet également de réduire le prix d'une unité.

Un mappage `ExpressionAttributeValues` est utilisé dans l'`UpdateExpression`. Les espaces réservés `:val1` et `:val2` sont remplacés lors de l'exécution par les valeurs réelles pour `Authors` et `Price`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price

UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
```

```
    expressionAttributeNames,
    expressionAttributeValues);
```

Spécification de paramètres facultatifs

Outre les paramètres obligatoires, vous pouvez également spécifier des paramètres facultatifs pour la méthode `updateItem`, y compris une condition à remplir afin que la mise à jour puisse se produire. Si la condition que vous spécifiez n'est pas remplie, AWS SDK pour Java lève une `ConditionalCheckFailedException`. Par exemple, l'extrait de code Java suivant redéfinit de façon conditionnelle le prix d'un élément livre sur la valeur 25. Il spécifie une `ConditionExpression` indiquant que le prix doit être mis à jour seulement s'il est de 20.

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Compteur atomique

`updateItem` vous permet d'implémenter un compteur atomique, où vous incrémentez ou décrémentez la valeur d'un attribut existant sans interférer avec d'autres demandes d'écriture. Pour incrémenter un compteur atomique, ajoutez une valeur numérique à un attribut existant de type `Number` à l'aide d'une `UpdateExpression` avec une action `set`.

L'extrait de code suivant illustre cette action, en incrémentant l'attribut `Quantity` d'une unité. Il montre également comment utiliser le paramètre `ExpressionAttributeNames` dans une `UpdateExpression`.

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);

UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

Suppression d'un élément

La méthode `deleteItem` supprime un élément d'une table. Vous devez fournir la clé primaire de l'élément que vous souhaitez supprimer.

Procédez comme suit :

1. Créez une instance du client `DynamoDB`.
2. Appelez la méthode `deleteItem` en fournissant la clé de l'élément que vous souhaitez supprimer.

L'extrait de code Java suivant illustre ces tâches.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Spécification de paramètres facultatifs

Vous pouvez spécifier des paramètres facultatifs pour `deleteItem`. Par exemple, l'extrait de code Java suivant comprend une `ConditionExpression`, indiquant qu'un élément livre dans

`ProductCatalog` peut être supprimé seulement si le livre n'est plus en publication. (L'attribut `InPublication` est false.)

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
    expressionAttributeValues);
```

Exemple : opérations CRUD à l'aide de l'API de document AWS SDK pour Java

L'exemple de code suivant illustre des opérations CRUD sur un élément Amazon DynamoDB. L'exemple crée un élément, le récupère, exécute plusieurs mises à jour et supprime finalement l'élément.

> **Note**
>
> Le kit SDK pour Java fournit également un modèle de persistance des objets qui vous permet de mapper vos classes côté client à des tables DynamoDB. Cette approche peut réduire la quantité de code que vous devez écrire. Pour en savoir plus, consultez Java 1.x : DynamoDBMapper.

> **Note**
>
> Cet exemple de code part du principe que vous avez déjà chargé des données dans DynamoDB pour votre compte en suivant les instructions de la section Création de tables et chargement de données pour des exemples de code dans DynamoDB.
> Pour obtenir des instructions pas à pas afin d'exécuter l'exemple suivant, consultez Exemples de code Java.

```
package com.amazonaws.codesamples.document;
```

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();

    }

    private static void createItems() {
```

```
        Table table = dynamoDB.getTable(tableName);
        try {

            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
120 Title")
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
Title")
                    .withString("ISBN", "121-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
```

```
            System.err.println(e.getMessage());
        }

    }

    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
"NewAttribute"))
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
ZZ").withString(":val2",
                                    "someValue"))
```

```
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {

        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
= :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
        }
```

```
    }

    private static void deleteItem() {

        Table table = dynamoDB.getTable(tableName);

        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
 120)
                    .withConditionExpression("#ip = :val").withNameMap(new
 NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
 false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);

            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

Exemple : opérations par lots à l'aide de l'API de document AWS SDK pour Java

Cette section fournit des exemples d'opérations d'écriture par lots et d'obtention par lots dans Amazon DynamoDB à l'aide de l'API de document AWS SDK pour Java.

> **Note**
>
> Le kit SDK pour Java fournit également un modèle de persistance des objets qui vous permet de mapper vos classes côté client à des tables DynamoDB. Cette approche peut réduire la quantité de code que vous devez écrire. Pour en savoir plus, consultez Java 1.x : DynamoDBMapper.

Rubriques

- Exemple : opérations d'écriture par lots à l'aide de l'API de document AWS SDK pour Java
- Exemple : opération d'obtention par lots à l'aide de l'API de document AWS SDK pour Java

Exemple : opérations d'écriture par lots à l'aide de l'API de document AWS SDK pour Java

L'exemple de code Java suivant exécute les opérations « put » et « delete » suivantes à l'aide de la méthode `batchWriteItem` :

- Insère un élément dans la table `Forum`.
- Insère un élément et supprime un élément de la table `Thread`.

Vous pouvez spécifier tout nombre de demandes put et delete sur une ou plusieurs tables lors de la création de votre demande d'écriture par lots. Cependant, `batchWriteItem` limite la taille d'une demande d'écriture par lots et le nombre d'opérations « put » et « delete » d'une même opération d'écriture par lots. Si votre demande dépasse ces limites, elle est rejetée. Si votre table n'a pas suffisamment de débit alloué pour traiter cette demande, les éléments non traités de la demande sont renvoyés dans la réponse.

L'exemple suivant vérifie la réponse pour voir si elle comporte des éléments de demande non traités. Si tel est le cas, une boucle est parcourue et la demande `batchWriteItem` est renvoyée avec les articles non traités de la demande. Si vous avez suivi les exemples de ce guide, vous devez déjà avoir créé les tables `Forum` et `Thread`. Vous pouvez également créer ces tables et charger des exemples de données par programmation. Pour en savoir plus, consultez Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour Java.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
```

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }

    private static void writeMultipleItemsBatchWrite() {
        try {

            // Add a new item to Forum
            TableWriteItems forumTableWriteItems = new
 TableWriteItems(forumTableName) // Forum
                    .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
 RDS").withNumber("Threads", 0));

            // Add a new item, and delete an existing item, from Thread
            // This table has a partition key and range key, so need to specify
            // both of them
            TableWriteItems threadTableWriteItems = new
 TableWriteItems(threadTableName)
                    .withItemsToPut(
                            new Item().withPrimaryKey("ForumName", "Amazon RDS",
 "Subject", "Amazon RDS Thread 1")
                                    .withString("Message", "ElastiCache Thread 1
 message")
                                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("cache", "in-memory"))))
```

```
                    .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
 "S3 Thread 100");

            System.out.println("Making the request.");
            BatchWriteItemOutcome outcome =
 dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);

            do {

                // Check for unprocessed keys which could happen if you exceed
                // provisioned throughput

                Map<String, List<WriteRequest>> unprocessedItems =
 outcome.getUnprocessedItems();

                if (outcome.getUnprocessedItems().size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Retrieving the unprocessed items");
                    outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
                }

            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

Exemple : opération d'obtention par lots à l'aide de l'API de document AWS SDK pour Java

L'exemple de code Java suivant récupère plusieurs éléments des tables `Forum` et `Thread` à l'aide de la méthode `batchGetItem`. La demande `BatchGetItemRequest` spécifie les noms de table et une liste des clés pour chaque élément à obtenir. L'exemple traite la réponse en imprimant les éléments récupérés.

> **Note**
>
> Cet exemple de code part du principe que vous avez déjà chargé des données dans DynamoDB pour votre compte en suivant les instructions de la section Création de tables et chargement de données pour des exemples de code dans DynamoDB.
> Pour obtenir des instructions pas à pas afin d'exécuter l'exemple suivant, consultez Exemples de code Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;

public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {
```

```
            TableKeysAndAttributes forumTableKeysAndAttributes = new
TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
"Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
"Subject", "Amazon DynamoDB",
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
"Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());
```

```
        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

Exemple : gestion d'attributs de type binaire à l'aide de l'API de document de AWS SDK pour Java

L'exemple de code Java suivant illustre la gestion des attributs de type binaire. L'exemple ajoute un élément à la table `Reply`. L'élément inclut un attribut de type binaire (`ExtendedMessage`) qui stocke les données compressées. Ensuite, l'exemple récupère l'élément et imprime toutes les valeurs d'attribut. À titre d'illustration, l'exemple compresse un exemple de flux et l'affecte à l'attribut `ExtendedMessage` à l'aide de la classe `GZIPOutputStream`. Lorsque le binary attribute est récupéré, il est décompressé à l'aide de la classe `GZIPInputStream`.

> **Note**
>
> Le kit SDK pour Java fournit également un modèle de persistance des objets qui vous permet de mapper vos classes côté client à des tables DynamoDB. Cette approche peut réduire la quantité de code que vous devez écrire. Pour en savoir plus, consultez Java 1.x : DynamoDBMapper.

Si vous avez suivi la section Création de tables et chargement de données pour des exemples de code dans DynamoDB, vous devez déjà avoir créé la table `Reply`. Vous pouvez également créer cette table par programmation. Pour en savoir plus, consultez Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour Java.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
```

```
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " +
e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id",
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId,
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);

        Item item = table.getItem(spec);

        // Uncompress the reply message and print
        String uncompressed =
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" +
" ReplyDateTime: "
```

```
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " +
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }

    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
                            // to the beginning
        return buffer;
    }

    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);
```

```
        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
        bais.close();

        return result;
    }
}
```

Utilisation d'élements : .NET

L'API de bas niveau AWS SDK pour .NET vous permet d'effectuer des opérations CRUD (création, lecture, mise à jour et suppression) classiques sur un élément dans une table. Voici les étapes courantes à suivre pour effectuer des opérations CRUD sur des données à l'aide de l'API de bas niveau .NET :

1. Créez une instance de la classe `AmazonDynamoDBClient` (le client).
2. Fournissez les paramètres obligatoires spécifiques à l'opération dans un objet de demande correspondant.

   Par exemple, utilisez l'objet de demande `PutItemRequest` lors du chargement d'un élément, et l'objet de demande `GetItemRequest` lors de l'extraction d'un élément existant.

   L'objet de demande vous permet de fournir les paramètres obligatoires et facultatifs.
3. Exécutez la méthode appropriée fournie par le client en transmettant l'objet de demande que vous avez créé à l'étape précédente.

   Le client `AmazonDynamoDBClient` fournit les méthodes `PutItem`, `GetItem`, `UpdateItem`, et `DeleteItem` pour les opérations CRUD.

Rubriques

- Insertion d'un élément
- Obtention d'un élément
- Mise à jour d'un élément
- Compteur atomique
- Suppression d'un élément
- Écriture par lots : insertion et suppression de plusieurs éléments
- Obtention par lots : obtention de plusieurs éléments
- Exemple : opérations CRUD à l'aide de l'API de bas niveau AWS SDK pour .NET
- Exemple : opérations par lots à l'aide de l'API de bas niveau AWS SDK pour .NET
- Exemple : gestion d'attributs de type binaire à l'aide de l'API de bas niveau AWS SDK pour .NET

Insertion d'un élément

La méthode `PutItem` charge un élément dans une table. Si l'élément existe, il remplace la totalité de l'élément.

> **Note**
>
> Au lieu de remplacer l'élément entier, vous pouvez mettre à jour uniquement des attributs spécifiques à l'aide de la méthode `UpdateItem`. Pour en savoir plus, consultez Mise à jour d'un élément.

Pour charger un élément à l'aide de l'API SDK .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires en créant une instance de la classe `PutItemRequest`.

   Pour insérer un élément, vous devez fournir le nom de la table et l'élément.
3. Appelez la méthode `PutItem` en fournissant l'objet `PutItemRequest` que vous avez créé à l'étape précédente.

L'exemple de code C# suivant illustre les étapes précédentes. L'exemple charge un élément dans la table `ProductCatalog`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

Dans l'exemple précédent, vous chargez un élément book (livre) qui possède les attributs `Id`, `Title`, `ISBN` et `Authors`. Notez que `Id` est un attribut de type numérique et que tous les autres attributs sont de type String (chaîne). Authors est un ensemble `String`.

Spécification de paramètres facultatifs

Vous pouvez également fournir des paramètres facultatifs à l'aide de l'objet `PutItemRequest`, comme illustré dans l'exemple C# suivant. L'exemple spécifie les paramètres facultatifs suivants :

- `ExpressionAttributeNames`, `ExpressionAttributeValues` et `ConditionExpression` spécifient que l'élément peut être remplacé seulement si l'élément existant possède l'attribut ISBN avec une valeur spécifique.
- Paramètre `ReturnValues` pour demander l'ancien élément dans la réponse.

Example

```
var request = new PutItemRequest
```

```
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                   { "ISBN", new AttributeValue { S = "444-4444444444" }},
                   { "Authors",
                     new AttributeValue { SS = new List<string>{"Author3"}}}
               },
    // Optional parameters.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

Pour en savoir plus, consultez PutItem.

Obtention d'un élément

La méthode `GetItem` extrait un élément.

> **Note**
>
> Vous pouvez extraire plusieurs éléments à l'aide de la méthode `BatchGetItem`. Pour en savoir plus, consultez Obtention par lots : obtention de plusieurs éléments.

Pour extraire un élément à l'aide de l'API AWS SDK pour .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires en créant une instance de la classe `GetItemRequest`.

   Pour obtenir un élément, vous devez fournir le nom de la table et la clé primaire de l'élément.

3. Appelez la méthode `GetItem` en fournissant l'objet `GetItemRequest` que vous avez créé à l'étape précédente.

L'exemple de code C# suivant illustre les étapes précédentes. L'exemple extrait un élément de la table `ProductCatalog`.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

Spécification de paramètres facultatifs

Vous pouvez également fournir des paramètres facultatifs à l'aide de l'objet `GetItemRequest`, comme illustré dans l'exemple C# suivant. L'exemple spécifie les paramètres facultatifs suivants :

- Paramètre `ProjectionExpression` pour spécifier les attributs à extraire.
- Paramètre `ConsistentRead` pour effectuer une lecture fortement cohérente. Pour en savoir plus sur la cohérence de lecture, consultez Cohérence en lecture DynamoDB.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
```

```
    // Optional parameters.
    ProjectionExpression = "Id, ISBN, Title, Authors",
    ConsistentRead = true
  };

  var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item;
```

Pour en savoir plus, consultez GetItem.

Mise à jour d'un élément

La méthode `UpdateItem` met à jour un élément existant s'il est présent. L'opération `UpdateItem` vous permet de mettre à jour des valeurs d'attribut existantes, d'ajouter des attributs ou d'en supprimer dans la collection existante. Si l'élément qui possède la clé primaire spécifiée est introuvable, l'opération ajoute un nouvel élément.

L'opération `UpdateItem` suit les directives suivantes :

- Si l'élément n'existe pas, `UpdateItem` ajoute un nouvel élément à l'aide de la clé primaire spécifiée dans l'entrée.
- Si l'élément existe, `UpdateItem` applique les mises à jour comme suit :
  - Remplace les valeurs d'attribut existantes par les valeurs figurant dans la mise à jour.
  - Si un attribut que vous fournissez en entrée n'existe pas, l'opération ajoute un nouvel attribut à l'élément.
  - Si l'attribut en entrée a la valeur null, elle supprime l'attribut s'il est présent.
  - Si vous utilisez `ADD` pour `Action`, vous pouvez ajouter des valeurs à un ensemble existant (ensemble de chaînes ou de nombres), ou bien ajouter (utiliser un nombre positif) ou soustraire (utiliser un nombre négatif) mathématiquement de façon à augmenter ou réduire la valeur de l'attribut numérique existant.

> **Note**
>
> L'opération `PutItem` peut également effectuer une mise à jour. Pour en savoir plus, consultez Insertion d'un élément. Par exemple, si vous appelez `PutItem` pour charger un

élément et que la clé primaire existe, l'opération `PutItem` remplace l'élément entier. Si l'élément existant comporte des attributs non spécifiés dans l'entrée, l'opération `PutItem` supprime ces attributs. Toutefois, l'opération `UpdateItem` ne met à jour que les attributs d'entrée spécifiés. Tous les autres attributs de cet élément demeurent inchangés.

Pour mettre à jour un élément à l'aide de l'API SDK .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires en créant une instance de la classe `UpdateItemRequest`.

   Il s'agit de l'objet de demande dans lequel vous décrivez toutes les mises à jour, comme l'ajout, la mise à jour ou la suppression d'attributs. Pour supprimer un attribut, spécifiez son nom avec la valeur null.
3. Appelez la méthode `UpdateItem` en fournissant l'objet `UpdateItemRequest` que vous avez créé à l'étape précédente.

L'exemple de code C# suivant illustre les étapes précédentes. L'exemple de code met à jour un élément book (livre) dans la table `ProductCatalog`. Il ajoute un nouvel auteur à la collection `Authors` et supprime l'attribut `ISBN` existant. Il permet également de réduire le prix d'une unité.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
```

```
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
var response = client.UpdateItem(request);
```

Spécification de paramètres facultatifs

Vous pouvez également fournir des paramètres facultatifs à l'aide de l'objet `UpdateItemRequest`, comme illustré dans l'exemple C# suivant. Il spécifie les paramètres facultatifs suivants :

- `ExpressionAttributeValues` et `ConditionExpression` pour spécifier que le prix peut être mis à jour seulement s'il est de 20,00.
- Paramètre `ReturnValues` pour demander l'élément mis à jour dans la réponse.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
```

```
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

Pour en savoir plus, consultez UpdateItem.

Compteur atomique

`updateItem` vous permet d'implémenter un compteur atomique, où vous incrémentez ou décrémentez la valeur d'un attribut existant sans interférer avec d'autres demandes d'écriture. Mettez à jour un compteur atomique à l'aide de `updateItem` avec un attribut de type `Number` dans le paramètre `UpdateExpression`, et ADD pour `Action`.

L'extrait de code suivant illustre cette action, en incrémentant l'attribut `Quantity` d'une unité.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

Suppression d'un élément

La méthode `DeleteItem` supprime un élément d'une table.

Pour supprimer un élément à l'aide de l'API SDK .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires en créant une instance de la classe `DeleteItemRequest`.

   Pour supprimer un élément, le nom de la table et la clé primaire de l'élément sont requis.
3. Appelez la méthode `DeleteItem` en fournissant l'objet `DeleteItemRequest` que vous avez créé à l'étape précédente.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

Spécification de paramètres facultatifs

Vous pouvez également fournir des paramètres facultatifs à l'aide de l'objet `DeleteItemRequest`, comme illustré dans l'exemple de code C# suivant. Il spécifie les paramètres facultatifs suivants :

- `ExpressionAttributeValues` et `ConditionExpression` pour spécifier que l'élément book (livre) peut être supprimé seulement s'il n'est plus dans la publication. (La valeur de l'attribut InPublication est false.)
- Paramètre `ReturnValues` pour demander l'élément supprimé dans la réponse.

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":inpub",new AttributeValue {BOOL = false}}
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

Pour en savoir plus, consultez DeleteItem.

Écriture par lots : insertion et suppression de plusieurs éléments

L'écriture par lots désigne l'insertion et la suppression de plusieurs éléments dans un lot. La méthode `BatchWriteItem` vous permet d'insérer et de supprimer plusieurs éléments d'une ou de plusieurs tables en un seul appel. Pour extraire plusieurs éléments à l'aide de l'API SDK .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Décrivez toutes les opérations d'insertion et de suppression en créant une instance de la classe `BatchWriteItemRequest`.
3. Appelez la méthode `BatchWriteItem` en fournissant l'objet `BatchWriteItemRequest` que vous avez créé à l'étape précédente.
4. Traitez la réponse. Vous devez vérifier si des éléments de demandes non traités ont été renvoyés dans la réponse. Cela peut se produire si vous atteignez le quota de débit alloué ou en cas d'autre erreur temporaire. En outre, DynamoDB limite la taille de demande et le nombre

d'opérations que vous pouvez spécifier dans une demande. Si vous dépassez ces limites, DynamoDB rejette la demande. Pour en savoir plus, consultez BatchWriteItem.

L'exemple de code C# suivant illustre les étapes précédentes. L'exemple crée un objet `BatchWriteItemRequest` pour effectuer les opérations d'écriture suivantes :

- Insérer un élément dans la table `Forum`
- Insérer et supprimer un élément dans la table `Thread`

Le code exécute `BatchWriteItem` pour effectuer une opération par lots.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
             }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
```

```
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
         },
         new WriteRequest
         {
            DeleteRequest = new DeleteRequest
            {
               Key = new Dictionary<string,AttributeValue>()
               {
                  { "ForumName", new AttributeValue { S = "Some forum name" } },
                  { "Subject", new AttributeValue { S = "Some subject" } }
               }
            }
         }
      }
   }
};
response = client.BatchWriteItem(request);
```

Pour obtenir un exemple pratique, consultez Exemple : opérations par lots à l'aide de l'API de bas niveau AWS SDK pour .NET.

Obtention par lots : obtention de plusieurs éléments

La méthode `BatchGetItem` vous permet de récupérer plusieurs éléments d'une ou de plusieurs tables.

> **Note**
>
> Vous pouvez récupérer un seul élément à l'aide de la méthode `GetItem`.

Pour extraire plusieurs éléments à l'aide de l'API AWS SDK pour .NET de bas niveau, procédez comme suit :

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires en créant une instance de la classe `BatchGetItemRequest`.

   Pour extraire plusieurs éléments, le nom de la table et une liste de valeurs de clé primaire sont requis.
3. Appelez la méthode `BatchGetItem` en fournissant l'objet `BatchGetItemRequest` que vous avez créé à l'étape précédente.
4. Traitez la réponse. Vous devez vérifier s'il existe des clés non traitées, ce qui peut se produire si vous atteignez le quota de débit alloué ou en cas d'autre erreur temporaire.

L'exemple de code C# suivant illustre les étapes précédentes. L'exemple extrait les éléments de deux tables : `Forum` et `Thread`. La demande spécifie deux éléments dans la table `Forum` et trois éléments dans la table `Thread`. La réponse inclut des éléments des deux tables. Le code montre comment vous pouvez traiter la réponse.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
```

```
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}
```

```
var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

Spécification de paramètres facultatifs

Vous pouvez également fournir des paramètres facultatifs à l'aide de l'objet `BatchGetItemRequest`, comme illustré dans l'exemple de code C# suivant. L'exemple extrait deux éléments de la table `Forum`. Il spécifie le paramètre facultatif suivant :

- Paramètre `ProjectionExpression` pour spécifier les attributs à extraire.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
```

```
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

Pour en savoir plus, consultez BatchGetItem.

Exemple : opérations CRUD à l'aide de l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant illustre des opérations CRUD sur un élément Amazon DynamoDB. L'exemple ajoute un élément à la table `ProductCatalog`, l'extrait, exécute diverses mises à jour et supprime finalement l'élément. Si vous n'avez pas créé cette table, vous pouvez également la créer par programmation. Pour en savoir plus, consultez Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour .NET.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  }},
                { "Title", new AttributeValue {
                      S = "Book 201 Title"
                  }},
                { "ISBN", new AttributeValue {
                      S = "11-11-11-11"
                  }},
                { "Authors", new AttributeValue {
                      SS = new List<string>{"Author1", "Author2" }
                  }},
                { "Price", new AttributeValue {
```

```
                N = "20.00"
              }},
            { "Dimensions", new AttributeValue {
                S = "8.5x11.0x.75"
              }},
            { "InPublication", new AttributeValue {
                BOOL = false
              } }
        }
        };
        client.PutItem(request);
    }

    private static void RetrieveItem()
    {
        var request = new GetItemRequest
        {
            TableName = tableName,
            Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
              } }
        },
            ProjectionExpression = "Id, ISBN, Title, Authors",
            ConsistentRead = true
        };
        var response = client.GetItem(request);

        // Check the response.
        var attributeList = response.Item; // attribute list in the response.
        Console.WriteLine("\nPrinting item after retrieving it ............");
        PrintItem(attributeList);
    }

    private static void UpdateMultipleAttributes()
    {
        var request = new UpdateItemRequest
        {
            Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
              } }
```

```
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
                 }},
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
update .............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
```

```
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
                // This updates price only if current price is 20.00.
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
            Console.WriteLine("\nPrinting item after updating price value
conditionally .............");
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
```

```
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);

            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted ............");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
```

```
}
```

Exemple : opérations par lots à l'aide de l'API de bas niveau AWS SDK pour .NET

Rubriques

- Exemple : opération d'écriture par lots à l'aide de l'API de bas niveau AWS SDK pour .NET
- Exemple : opération d'obtention par lots à l'aide de l'API de bas niveau AWS SDK pour .NET

Cette section fournit des exemples d'opérations par lots, écriture par lots et obtention par lots, qu'Amazon DynamoDB prend en charge.

Exemple : opération d'écriture par lots à l'aide de l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant exécute les opérations d'insertion et de suppression suivantes à l'aide de la méthode `BatchWriteItem` :

- Insère un élément dans la table `Forum`.
- Insère un élément et supprime un élément de la table `Thread`.

Vous pouvez spécifier tout nombre de demandes put et delete sur une ou plusieurs tables lors de la création de votre demande d'écriture par lots. Cependant, DynamoDB `BatchWriteItem` limite la taille d'une demande d'écriture par lots et le nombre d'opérations d'insertion et de suppression dans une même opération d'écriture par lots. Pour en savoir plus, consultez BatchWriteItem. Si votre demande dépasse ces limites, elle est rejetée. Si votre table n'a pas suffisamment de débit alloué pour traiter cette demande, les éléments non traités de la demande sont renvoyés dans la réponse.

L'exemple suivant vérifie la réponse pour voir si elle comporte des éléments de demande non traités. Si tel est le cas, une boucle est parcourue et la demande `BatchWriteItem` est renvoyée avec les articles non traités de la demande. Vous pouvez également créer ces exemples de tables et charger des exemples de données par programmation. Pour en savoir plus, consultez Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour .NET.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
```

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
```

```
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    },
                    {
                        table2Name, new List<WriteRequest>
                        {
                            new WriteRequest
                            {
                                PutRequest = new PutRequest
                                {
                                    Item = new Dictionary<string, AttributeValue>
                                    {
                                        { "ForumName", new AttributeValue {
                                              S = "S3 forum"
                                        } },
                                        { "Subject", new AttributeValue {
                                              S = "My sample question"
                                        } },
                                        { "Message", new AttributeValue {
                                              S = "Message Text."
                                        } },
                                        { "KeywordTags", new AttributeValue {
                                              SS = new List<string> { "S3", "Bucket" }
                                        } }
                                    }
                                }
                            },
                            new WriteRequest
                            {
                                // For the operation to delete an item, if you provide a
primary key value
                                // that does not exist in the table, there is no error, it
is just a no-op.
                                DeleteRequest = new DeleteRequest
                                {
                                    Key = new Dictionary<string, AttributeValue>()
                                    {
                                        { "ForumName",  new AttributeValue {
                                              S = "Some partition key value"
```

```
                                    } },
                                  { "Subject", new AttributeValue {
                                        S = "Some sort key value"
                                    } }
                                }
                            }
                        }
                    }
                }
            }
            };

            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
            BatchWriteItemResponse response;

            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.

                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();
```

```
                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

Exemple : opération d'obtention par lots à l'aide de l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant extrait plusieurs éléments des tables `Forum` et `Thread` dans Amazon DynamoDB à l'aide de la méthode `BatchGetItem`. La demande `BatchGetItemRequest` spécifie les noms de table et une liste de clés primaires pour chaque table. L'exemple traite la réponse en imprimant les éléments récupérés.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
```

```
            Console.ReadLine();
        }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void RetrieveMultipleItemsBatchGet()
    {
        var request = new BatchGetItemRequest
        {
            RequestItems = new Dictionary<string, KeysAndAttributes>()
        {
            { table1Name,
              new KeysAndAttributes
              {
                  Keys = new List<Dictionary<string, AttributeValue> >()
                  {
                      new Dictionary<string, AttributeValue>()
                      {
                          { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                      } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "Name", new AttributeValue {
                            S = "Amazon S3"
                      } }
                        }
                  }
              }},
            {
                table2Name,
                new KeysAndAttributes
                {
                    Keys = new List<Dictionary<string, AttributeValue> >()
                    {
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                                } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 1"
```

```
                                } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchGetItem(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
```

```
                }

                // Any unprocessed keys? could happen if you exceed
 ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
 response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

Exemple : gestion d'attributs de type binaire à l'aide de l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant illustre la gestion des attributs de type binaire. L'exemple ajoute un élément à la table `Reply`. L'élément inclut un attribut de type binaire (`ExtendedMessage`) qui stocke les données compressées. Ensuite, l'exemple récupère l'élément et imprime toutes les valeurs d'attribut. A titre d'illustration, l'exemple compresse un exemple de flux et l'affecte à l'attribut `ExtendedMessage` à l'aide de la classe `GZipStream`, puis le décompresse lors de l'affichage de la valeur de l'attribut.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
```

```
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
message to compress.");
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  }},
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  }},
                { "Subject", new AttributeValue {
                      S = "Binary type "
                  }},
                { "Message", new AttributeValue {
                      S = "Some message about the binary type"
                  }},
                { "ExtendedMessage", new AttributeValue {
                      B = compressedMessage
                  }}
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem(string partitionKey, string sortKey)
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
```

```
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            },
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");

            PrintItem(attributeList);
        }

        private static void DeleteItem(string partitionKey, string sortKey)
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
```

```
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
 "]")
                    );
            }
            Console.WriteLine("*************************************************");
        }

        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
 CompressionMode.Compress, true))
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
 CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Amélioration de l'accès aux données avec les index secondaires dans DynamoDB

Amazon DynamoDB fournit un accès rapide aux éléments d'une table en spécifiant des valeurs de clé principale. Cependant, il serait utile pour de nombreuses applications de disposer d'une ou de plusieurs clés secondaires (ou alternatives) permettant un accès efficace aux données présentant d'autres attributs que la clé primaire. Pour ce faire, vous pouvez créer un ou plusieurs index secondaires pour une table, et émettre des demandes `Query` ou `Scan` sur ces index.

Un index secondaire est une structure de données qui contient un sous-ensemble d'attributs d'une table, ainsi qu'une autre clé pour prendre en charge des opérations `Query`. Vous pouvez récupérer les données de l'index à l'aide d'un `Query`, globalement de la même manière que vous utilisez `Query` avec une table. Une table peut avoir plusieurs index secondaires, ce qui permet à vos applications d'accéder à de nombreux modèles de requête différents.

> **Note**
>
> Vous pouvez également `Scan` un index, globalement de la même manière que vous le feriez pour `Scan` une table.
> L'accès intercompte pour les opérations d'analyse d'index secondaires n'est actuellement pas pris en charge par les politiques basées sur les ressources.

Chaque index secondaire est associé à une table dont il extrait ses données. Il s'agit de la table de base de l'index. Lorsque vous créez un index, vous définissez une autre clé pour cet index (clé de partition et clé de tri). Vous définissez également les attributs qui doivent être projetés, ou copiés de la table de base vers l'index. DynamoDB copie ces attributs dans l'index, ainsi que les attributs de clé primaire de la table de base. Vous pouvez ensuite interroger ou analyser l'index de la même manière que vous procéderiez avec une table.

DynamoDB grère automatiquement chaque index secondaire . Lorsque vous ajoutez, modifiez ou supprimez des éléments dans la table de base, tous les index de cette table sont également mis à jour pour refléter ces modifications.

DynamoDB prend en charge deux types d'index secondaires :

- Index secondaire global – Index avec une clé de partition et une clé de tri pouvant différer de celles de la table de base. Un index secondaire global est considéré comme « global », car les requêtes

sur l'index peuvent couvrir toutes les données de la table de base, sur toutes les partitions. Un index secondaire global est stocké dans son propre espace de partition à l'écart de la table de base, et est mis à l'échelle séparément de celle-ci.

- Index secondaire local – Index avec la même clé de partition que la table de base, mais une clé de tri différente. Un index secondaire est « local » dans la mesure où la portée de chacune de ses partitions correspond à une partition de la table de base ayant la même valeur de clé de partition.

Pour une comparaison des index secondaires globaux et des index secondaires locaux, regardez cette vidéo.

Bien choisir entre un index secondaire global et un index secondaire local

Rubriques

- Utilisation d'index secondaires globaux dans DynamoDB
- Index locaux secondaires

Vous devez tenir compte des exigences de votre application lorsque vous déterminez le type d'index à utiliser. Le tableau suivant montre les principales différences entre un index secondaire global et un index secondaire local.

Caractéristiques	GSI	Index secondaire local
Schéma de clé	La clé primaire d'un index secondaire global peut être simple (clé de partition) ou composite (clé de partition et clé de tri).	La clé primaire d'un index secondaire local doit être composite (clé de partition et clé de tri).
Attributs de clé	La clé de partition d'index et la clé de tri (le cas échéant) peuvent être n'importe quels attributs de table de base de type chaîne, nombre ou binaire.	La clé de partition de l'index est le même attribut que la clé de partition de la table de base. La clé de tri peut être n'importe quel attribut de la table de base, de type chaîne, nombre ou binaire.

Caractéristiques	GSI	Index secondaire local
Restrictions de taille par valeur de clé de partition	Les index secondaires globaux ne font l'objet d'aucune restriction de taille .	Pour chaque valeur de clé de partition, la taille totale de tous les éléments indexés doit être inférieure ou égale à 10 Go.
Opérations d'index en ligne	Vous pouvez créer des index secondaires globaux au moment où vous créez une table. Vous pouvez ajouter un index secondaire global à une table ou en supprimer un. Pour en savoir plus, consultez Gestion des index secondaires globaux dans DynamoDB.	Vous créez des index secondaires locaux au moment où vous créez une table. Vous ne pouvez ni ajouter un index secondair e local à une table, ni en supprimer un.
Requêtes et partitions	Un index secondaire global vous permet d'interroger une table entière, toutes partitions incluses.	Un index secondaire local vous permet d'interroger une seule partition en spécifiant la valeur de clé de partition dans la requête.
Cohérence en lecture	Les requêtes sur les index secondaires globaux ne prennent en charge que la cohérence éventuelle.	Lorsque vous interrogez un index secondaire local, vous avez le choix entre la cohérence éventuelle ou la cohérence forte.

Caractéristiques	GSI	Index secondaire local
Consommation de débit alloué	Chaque index secondaire global a ses propres paramètres de débit approvisionné pour les activités de lecture et d'écriture. Les requêtes ou analyses sur un index secondaire global consomment des unités de capacité de l'index, pas de la table de base. Il en va de même pour des mises à jour d'index secondaire global résultant d'écritures dans la table. Un index secondaire global associé à des tables globales consomme des unités de capacité d'écriture.	Les requêtes ou analyses sur un index secondaire local consomment des unités de capacité de lecture de la table de base. Lorsque vous écrivez des données dans une table, ses index secondaires locaux sont mis à jour, ce qui entraîne une consommation des unités de capacité d'écriture à partir de la table de base. Un index secondaire local associé à des tables globales consomme des unités de capacité d'écriture répliquées.
Attributs projetés	Les requêtes ou analyses d'index secondaire global ne vous permettent d'interroger que les attributs projetés dans l'index. DynamoDB n'extrait aucun attribut de la table.	Si vous interrogez ou analysez un index secondaire local, vous pouvez demander des attributs qui ne sont pas projetés dans l'index. DynamoDB extrait automatiquement ces attributs de la table.

Si vous souhaitez créer plus d'une table avec des index secondaires, vous devez le faire de manière séquentielle. Par exemple, vous créez la première table et attendez qu'elle devienne `ACTIVE`, vous créez la table suivante et attendez qu'elle devienne `ACTIVE`, et ainsi de suite. Si vous essayez de créer simultanément plus d'une table avec un index secondaire, DynamoDB renvoie une `LimitExceededException`.

Chaque index secondaire utilise la même classe de table et le même mode de capacité que la table de base à laquelle il est associé. Pour chaque index secondaire, vous devez spécifier les informations suivantes :

- Type d'index à créer : index secondaire global ou un index secondaire local.
- Un nom pour l'index. Les règles de dénomination pour les index sont identiques à celles pour les tables, comme indiqué dans Quotas dans Amazon DynamoDB. Le nom doit être unique pour la table de base à laquelle il est associé, mais vous pouvez utiliser le même nom pour des index qui sont associés à d'autres tables de base.
- Le schéma de clés pour l'index. Chaque attribut du schéma de clé d'index doit être un attribut de niveau supérieur de type `String`, `Number` ou `Binary`. D'autres types de données, y compris les documents et les jeux, ne sont pas autorisés. Les autres exigences pour le schéma de clé dépendent du type d'index :
  - Pour un index secondaire global, la clé de partition peut être n'importe quel attribut scalaire de la table de base. Une clé de tri est facultative et peut également être n'importe quel attribut scalaire de la table de base.
  - Pour un index secondaire local, la clé de partition doit être la même que celle de la table de base, et la clé de tri doit être un attribut de la table de base autre qu'une clé.
- Attributs supplémentaires, le cas échéant, à projeter de la table de base vers l'index. Ces attributs sont en plus des attributs clés de la table, qui sont automatiquement projetés dans chaque index. Vous pouvez projeter des attributs de n'importe quel type de données, y compris scalaires, documents et jeux.
- Les paramètres de débit alloué pour l'index, si nécessaire :
  - Pour un index secondaire global, vous devez spécifier des paramètres d'unité de capacité de lecture et d'écriture. Ces paramètres de débit alloué sont indépendants des paramètres de la table de base.
  - Pour un index secondaire local, vous n'avez pas besoin de spécifier des paramètres d'unité de capacité de lecture et d'écriture. Toute opération de lecture et d'écriture sur un index secondaire local tire les paramètres de débit approvisionné de sa table de base.

Pour bénéficier d'une flexibilité de requête maximum, vous pouvez créer jusqu'à 20 index secondaires globaux (quota par défaut) et 5 index secondaires locaux par table.

Pour obtenir la liste détaillée des index secondaires sur une table, utilisez l'opération `DescribeTable`. L'opération `DescribeTable` renvoie le nom, la taille de stockage et le nombre

d'éléments pour chaque index secondaire sur la table. Ces valeurs ne sont pas mises à jour en temps réel, mais elles sont actualisées environ toutes les six heures.

Vous pouvez accéder aux données dans un index secondaire à l'aide de l'opération `Query` ou `Scan`. Vous devez spécifier le nom de la table de base et le nom de l'index que vous souhaitez utiliser, les attributs à renvoyer dans les résultats, ainsi que tous les filtres ou expressions de condition que vous souhaitez appliquer. DynamoDB peut renvoyer les résultats dans l'ordre croissant ou décroissant.

Lorsque vous supprimez une table, tous les index associés à cette table sont également supprimés.

Pour connaître les bonnes pratiques, consultez Bonnes pratiques d'utilisation d'index secondaires dans DynamoDB.

Utilisation d'index secondaires globaux dans DynamoDB

Certaines applications peuvent avoir besoin d'exécuter de nombreux types de requêtes, à l'aide de divers attributs comme critères de requête. Pour prendre en charge ces exigences, vous pouvez créer un ou plusieurs index secondaires globaux et émettre des demandes `Query` sur ces index dans Amazon DynamoDB.

Rubriques

- Étape 6 : Utiliser un index secondaire global
- Projections d'attribut
- Schéma de clé à attributs multiples
- Lecture de données à partir d'un index secondaire global
- Synchronisation des données entre les tables et les index secondaires globaux
- Classes de tables avec index secondaire global
- Considérations sur le débit alloué pour les index secondaires globaux
- Considérations relatives au stockage pour les index secondaires globaux
- Motifs de design
- Gestion des index secondaires globaux dans DynamoDB
- Détection et correction des violations de clé d'index dans DynamoDB
- Utilisation d'index secondaires globaux : Java
- Utilisation d'index secondaires globaux : .NET
- Gestion des index secondaires globaux dans DynamoDB à l'aide de l'AWS CLI

Étape 6 : Utiliser un index secondaire global

À des fins d'illustration, considérons une table nommée `GameScores` qui assure le suivi des utilisateurs et enregistre les scores d'une application de jeux pour appareils mobiles. Chaque élément dans `GameScores` est identifié par une clé de partition (`UserId`) et une clé de tri (`GameTitle`). Le schéma suivant illustre la façon dont les éléments de la table seront organisés. (Tous les attributs ne sont pas affichés.)

GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Supposons maintenant que vous vouliez écrire une application de classement pour afficher les meilleurs scores de chaque partie. Une requête ayant spécifié les attributs clé (`UserId` et `GameTitle`) serait très efficace. Cependant, si l'application a besoin d'extraire les données depuis `GameScores` en fonction de `GameTitle` uniquement, elle nécessiterait l'utilisation d'une opération `Scan`. Lorsque plusieurs éléments sont ajoutés à la table, les analyses de toutes les données deviennent lentes et inefficaces. Cela rend difficile la réponse aux questions suivantes :

- Quel est le meilleur score jamais enregistré pour le jeu Meteor Blasters ?
- Quel utilisateur a le score le plus élevé pour Galaxy Invaders ?
- Quel est le taux plus élevé du nombre de victoires par rapport au nombre de pertes ?

Afin d'accélérer les requêtes sur des attributs autres que de clé, vous pouvez créer un index secondaire global. Un index secondaire global contient une sélection d'attributs de la table de base, mais ils sont organisés selon une clé primaire différente de celle de la table. La clé d'index n'a pas besoin d'avoir l'un des attributs de clé de la table. Elle n'a même pas besoin d'avoir le même schéma de clé qu'une table.

Par exemple, vous pouvez créer un index secondaire global nommé `GameTitleIndex`, avec une clé de partition `GameTitle` et une clé de tri `TopScore`. Comme les attributs de la clé primaire de la table de base sont toujours projetés sur un index, l'attribut `UserId` est également présent. Le schéma suivant illustre ce à quoi l'index `GameTitleIndex` doit ressembler.

GameTitleIndex

GameTitle	*TopScore*	*UserId*
"Alien Adventure"	*192*	*"102"*
"Attack Ships"	*3*	*"103"*
"Galaxy Invaders"	*0*	*"102"*
"Galaxy Invaders"	*2317*	*"103"*
"Galaxy Invaders"	*5842*	*"101"*
"Meteor Blasters"	*723*	*"103"*
"Meteor Blasters"	*1000*	*"101"*
"Starship X"	*24*	*"101"*
"Starship X"	*42*	*"103"*
...	...	...

Maintenant, vous pouvez interroger `GameTitleIndex` et obtenir facilement les scores de Meteor Blasters. Les résultats sont triés sur les valeurs de la clé de tri, `TopScore`. Si vous définissez le paramètre `ScanIndexForward` sur la valeur false, les résultats sont retournés par ordre décroissant, de telle sorte que le meilleur score est retourné en premier.

Chaque index secondaire global doit avoir une clé de partition et peut avoir une clé de tri facultative. Le schéma de la clé d'index peut être différent de celui de la table de base. Vous pouvez disposer

d'une table avec une clé primaire simple (clé de partition), et créer un index secondaire global avec une clé primaire composite (clé de partition et clé de tri), ou inversement. Les attributs de clé d'index peuvent se composer de n'importe lequel des attributs `String`, `Number` ou `Binary` de niveau supérieur de la table de base. Les autres types scalaires, types de document et types d'ensemble ne sont pas autorisés.

Vous pouvez projeter d'autres attributs de table de base sur l'index si vous le souhaitez. Lorsque vous interrogez l'index, DynamoDB peut extraire ces attributs projetés de façon efficace. Cependant, les interrogations d'index secondaire global ne peuvent pas extraire d'attributs de la table de base. Par exemple, si vous interrogez `GameTitleIndex` comme illustré dans le diagramme précédent, la requête ne peut pas accéder aux attributs non-clé autres que `TopScore` (même si les attributs de clé `GameTitle` et `UserId` sont automatiquement projetés).

Dans une table DynamoDB, chaque valeur de clé doit être unique. En revanche, les valeurs de clé dans un index secondaire global n'ont pas besoin d'être uniques. À des fins d'illustration, supposons qu'un jeu nommé Comet Quest soit particulièrement difficile, avec de nombreux utilisateurs qui tentent en vain d'obtenir un score supérieur à zéro. Les quelques données suivantes représentent ce comportement.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Lorsque de l'ajout de ces données à la table `GameScores`, DynamoDB les propage vers `GameTitleIndex`. Si nous interrogeons alors l'index à l'aide de Comet Quest pour `GameTitle` et de 0 pour `TopScore`, les données suivantes sont renvoyées.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Seuls les éléments avec les valeurs de clé spécifiée apparaissent dans la réponse. Au sein de cet ensemble de données, les éléments ne figurent dans aucun ordre particulier.

Un index secondaire global ne suit que les éléments de données où ses attributs clés existent réellement. Par exemple, supposons que vous ayez ajouté un nouvel élément à la table `GameScores`, mais que vous n'ayez fourni que les attributs de clé primaires requis.

UserId	GameTitle
400	Comet Quest

Étant donné que vous n'avez pas spécifié l'attribut `TopScore`, DynamoDB ne propage pas cet élément vers `GameTitleIndex`. Par conséquent, si vous interrogiez `GameScores` pour tous les éléments Comet Quest, vous obtiendriez les quatre éléments suivants.

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Une requête similaire sur `GameTitleIndex` renvoie toujours trois éléments, plutôt que quatre. Cela est dû au fait que l'élément avec `TopScore` non existant n'est pas propagé sur l'index.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Projections d'attribut

Une projection est l'ensemble d'attributs copié à partir d'une table dans un index secondaire. Les clés de partition et de tri de la table sont toujours projetées dans l'index. Vous pouvez projeter d'autres attributs en fonction des exigences de requête de votre application. Lorsque vous interrogez un index,

Amazon DynamoDB peut accéder à n'importe quel attribut dans la projection, comme s'il se trouvait dans une table.

Lorsque vous créez un index secondaire, vous devez spécifier les attributs qui seront projetés dans celui-ci. Pour cela, DynamoDB propose trois options :

- KEYS_ONLY – Chaque élément de l'index se compose uniquement des valeurs de clé de partition et de tri de la table, ainsi que des valeurs de clé d'index. L'option `KEYS_ONLY` produit l'index secondaire le plus petit possible.
- INCLUDE – En plus des attributs décrits dans `KEYS_ONLY`, l'index secondaire inclut des attributs autres que de clé, que vous spécifiez.
- ALL – L'index secondaire inclut tous les attributs de la table source. Toutes les données de la table étant dupliquées dans l'index, une projection `ALL` produit l'index secondaire le plus grand possible.

Dans le précédent diagramme, `GameTitleIndex` n'a qu'un seul attribut projeté : `UserId`. Par conséquent, si une application peut déterminer efficacement le `UserId` des joueurs avec les scores les plus élevés pour chaque jeu à l'aide de `GameTitle` et de `TopScore` dans des requêtes, elle peut déterminer efficacement le taux le plus élevé de victoires par rapport au nombre de défaites pour ces joueurs. Pour ce faire, elle doit exécuter une requête supplémentaire sur la table de base afin d'extraire les victoires et les défaites pour chacun des joueurs aux scores les plus élevés. Une façon plus efficace de prendre en charge les requêtes sur ces données serait de projeter ces attributs de la table de base vers l'index secondaire global, comme illustré dans ce diagramme.

GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
"Alien Adventure"	192	"102"	32	192
"Attack Ships"	3	"103"	1	8
"Galaxy Invaders"	0	"102"	0	5
"Galaxy Invaders"	2317	"103"	40	3
"Galaxy Invaders"	5842	"101"	21	72
"Meteor Blasters"	723	"103"	22	12
"Meteor Blasters"	1000	"101"	12	3
"Starship X"	24	"101"	4	9
"Starship X"	42	"103"	4	19
...	...	...	...	...

Comme les attributs non clés `Wins` et`Losses` sont projetés dans l'index, une application peut déterminer le rapport entre le nombre de victoires et le nombre de défaites de n'importe quel jeu, ou pour toute combinaison d'ID d'utilisateur et de jeu.

Lorsque vous choisissez les attributs à projeter sur un index secondaire global, vous devez prendre en compte le compromis entre les coûts de débit approvisionné et les coûts de stockage :

- Si vous n'avez besoin d'accéder qu'à quelques attributs avec la plus faible latence possible, envisagez de projeter ces seuls attributs dans un index secondaire global. Plus l'index est petit, moins les coûts d'écriture et de stockage sont élevés.
- Si votre application accède fréquemment à des attributs autres que de clé, vous devez envisager de projeter ceux-ci dans un index secondaire global. Les coûts de stockage supplémentaires de l'index secondaire global compensent le coût d'exécution d'analyses de table fréquentes.
- Si vous avez besoin d'accéder fréquemment à la plupart des attributs autres que de clé, vous pouvez projeter ceux-ci (voire la table de base toute entière) dans un index secondaire global.

Cela vous offre une flexibilité maximale. Cependant, vos coûts de stockage augmenteront, voire doubleront.

- Si votre application doit interroger peu fréquemment une table, mais doit exécuter un grand nombre d'écritures ou de mises à jour des données de la table, pensez à projeter `KEYS_ONLY`. L'index secondaire global sera d'une taille minimale, mais continuera d'être disponible chaque fois que ce sera nécessaire pour une activité de requête.

Schéma de clé à attributs multiples

Les index secondaires globaux prennent en charge les clés à attributs multiples, ce qui vous permet de composer des clés de partition et de trier les clés à partir de plusieurs attributs. Avec les clés à attributs multiples, vous pouvez créer une clé de partition à partir de quatre attributs et une clé de tri à partir de quatre attributs, pour un total de huit attributs par schéma de clé.

Les clés à attributs multiples simplifient votre modèle de données en éliminant le besoin de concaténer manuellement les attributs dans des clés synthétiques. Au lieu de créer des chaînes composites`TOURNAMENT#WINTER2024#REGION#NA-EAST`, vous pouvez utiliser directement les attributs naturels de votre modèle de domaine. DynamoDB gère automatiquement la logique des clés composites, en hachant plusieurs attributs de clé de partition pour la distribution des données et en maintenant un ordre de tri hiérarchique entre plusieurs attributs de clé de tri.

Par exemple, imaginez un système de tournois de jeu dans lequel vous souhaitez organiser les matchs par tournoi et par région. Avec les clés à attributs multiples, vous pouvez définir votre clé de partition sous la forme de deux attributs distincts : `tournamentId` et`region`. De même, vous pouvez définir votre clé de tri à l'aide de plusieurs attributs tels que `roundbracket`, et `matchId` pour créer une hiérarchie naturelle. Cette approche permet de saisir vos données et de garder votre code propre, sans manipulation de chaînes ni analyse syntaxique.

Lorsque vous interrogez un index secondaire global avec des clés à attributs multiples, vous devez spécifier tous les attributs de clé de partition en utilisant des conditions d'égalité. Pour trier les attributs clés, vous pouvez les interroger left-to-right dans l'ordre dans lequel ils sont définis dans le schéma de clé. Cela signifie que vous pouvez interroger le premier attribut clé de tri seul, les deux premiers attributs ensemble ou tous les attributs ensemble, mais vous ne pouvez pas ignorer les attributs situés au milieu. Les conditions d'inégalité telles que > <`BETWEEN`,, ou `begins_with()` doivent être la dernière condition de votre requête.

Les clés à attributs multiples fonctionnent particulièrement bien lors de la création d'index secondaires globaux sur des tables existantes. Vous pouvez utiliser des attributs qui existent déjà

dans votre table sans avoir à ajouter des clés synthétiques à vos données. Cela permet d'ajouter facilement de nouveaux modèles de requête à votre application en créant des index qui réorganisent vos données à l'aide de différentes combinaisons d'attributs.

Chaque attribut d'une clé à attributs multiples peut avoir son propre type de données : `String` (S), `Number` (N) ou `Binary` (B). Lorsque vous choisissez des types de données, considérez que `Number` les attributs sont triés numériquement sans nécessiter de remplissage nul, tandis que les `String` attributs sont triés de manière lexicographique. Par exemple, si vous utilisez un `Number` type pour un attribut de score, les valeurs 5, 50, 500 et 1000 sont triées par ordre numérique naturel. Les mêmes valeurs que `String` type seraient triées comme « 1000 », « 5 », « 50 », « 500 », sauf si vous les complétez avec des zéros en tête.

Lorsque vous concevez des clés à attributs multiples, classez vos attributs du plus général au plus spécifique. Pour les clés de partition, combinez les attributs qui sont toujours interrogés ensemble et qui assurent une bonne distribution des données. Pour les clés de tri, placez les attributs fréquemment demandés en premier dans la hiérarchie afin d'optimiser la flexibilité des requêtes. Cet ordre vous permet d'effectuer des requêtes à n'importe quel niveau de granularité correspondant à vos habitudes d'accès.

Consultez le Clés à attributs multiples pour des exemples de mise en œuvre.

Lecture de données à partir d’un index secondaire global

Vous pouvez extraire des éléments d’un index secondaire global à l’aide des opérations `Query` et `Scan`. Vous ne pouvez pas utiliser les opérations `GetItem` et `BatchGetItem` sur un index secondaire global.

Interrogation d’un index secondaire global

Vous pouvez utiliser l’opération `Query` pour accéder à un ou plusieurs éléments dans un index secondaire global. La requête doit spécifier le nom de la table de base et le nom de l’index que vous souhaitez utiliser, les attributs à renvoyer dans les résultats de la requête, ainsi que toutes les conditions de requête que vous souhaitez appliquer. DynamoDB peut renvoyer les résultats dans l’ordre croissant ou décroissant.

Prenez en compte les données suivantes retournées à partir d’une opération `Query` qui demande les données de jeu pour une application de classement.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
```

```
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

Dans cette requête :

- DynamoDB y GameTitleIndexaccède à l'aide de la clé de partition pour localiser GameTitleles éléments d'index des Meteor Blasters. Tous les éléments d'index avec cette clé sont stockés les uns à côté des autres pour une récupération rapide.
- Dans ce jeu, DynamoDB utilise l'index pour accéder à tous les IDs utilisateurs et aux meilleurs scores de ce jeu.
- Les résultats sont retournés, triés par ordre décroissant, car le paramètre `ScanIndexForward` a la valeur false.

Interrogation d'un index secondaire global

Vous pouvez utiliser l'opération `Scan` pour extraire toutes les données d'un index secondaire global. Vous devez fournir le nom de la table de base et le nom de l'index dans la demande. Avec une opération `Scan`, DynamoDB lit toutes les données de l'index et les renvoie à l'application. Vous pouvez également demander que seules quelques données soient retournées, et que les données restantes soient ignorées. Pour ce faire, utilisez le paramètre `FilterExpression` de l'opération `Scan`. Pour de plus amples informations, veuillez consulter Filtrer les expressions à des fins d'analyse.

Synchronisation des données entre les tables et les index secondaires globaux

DynamoDB synchronise automatiquement chaque index secondaire global avec sa table de base. Quand une application écrit ou supprime des éléments dans une table, tout index secondaire global sur cette table est mis à jour de manière asynchrone, à l'aide d'un modèle éventuellement cohérent. Les applications n'écrivent jamais directement sur un index. Cependant, il est important de comprendre les implications de la façon dont DynamoDB gère ces index.

Les index secondaires globaux héritent du mode read/write capacité de la table de base. Pour de plus amples informations, veuillez consulter Considérations relatives au changement de mode de capacité dans DynamoDB.

Lorsque vous créez un index secondaire global, vous spécifiez un ou plusieurs attributs de clé d'index et leurs types de données. Cela signifie que chaque fois que vous écrivez un élément sur la table de base, les types de données de ces attributs doivent correspondre aux types de données du schéma de la clé d'index. Dans le cas de `GameTitleIndex`, la clé de partition `GameTitle` de l'index est définie comme type de données `String`. La clé de tri `TopScore` de l'index est de type `Number`. Si vous essayez d'ajouter un élément à la table `GameScores` et spécifiez un type de données différent pour `GameTitle` ou pour `TopScore`, DynamoDB renvoie une `ValidationException` en raison de la discordance des types de données.

Lorsque vous insérez ou supprimez des éléments dans une table, les index secondaires globaux sont mis à jour selon une façon éventuellement cohérente. Les modifications apportées aux données de la table sont propagées sur les index secondaires globaux en une fraction de seconde, dans des conditions normales. Cependant, dans certains scénarios de défaillance peu probables, les retards de propagation peuvent être plus longs. Pour cette raison, vos applications ont besoin d'anticiper et de gérer les situations où une requête sur un index secondaire global renvoie des résultats qui ne sont pas à jour.

Si vous écrivez un élément dans une table, vous n'avez pas à spécifier les attributs pour une clé de tri d'index secondaire global. Si vous utilisez `GameTitleIndex` comme exemple, vous n'avez pas besoin de spécifier une valeur pour l'attribut `TopScore` pour pouvoir écrire un nouvel élément sur la table `GameScores`. Dans ce cas, DynamoDB n'écrit aucune donnée dans l'index pour cet élément particulier.

Une table avec de nombreux index secondaires globaux s'expose à des coûts plus élevés pour l'activité d'écriture qu'une table ayant moins d'index. Pour de plus amples informations, veuillez consulter Considérations sur le débit alloué pour les index secondaires globaux.

Classes de tables avec index secondaire global

Un index secondaire global utilisera toujours la même classe de tables que sa table de base. Chaque fois qu'un nouvel index secondaire global est ajouté à une table, le nouvel index utilise la même classe de tables que sa table de base. Lorsque la classe de tables d'une table est mise à jour, tous les index secondaires globaux associés sont également mis à jour.

Considérations sur le débit alloué pour les index secondaires globaux

Lorsque vous créez un index secondaire global sur une table en mode approvisionné, vous devez spécifier des unités de capacité de lecture et d'écriture pour la charge de travail prévue sur cet

index. Les paramètres de débit approvisionné d'un index secondaire global sont distincts de ceux de sa table de base. Une opération `Query` sur un index secondaire global consomme les unités de capacité de lecture de l'index, pas de la table de base. Lorsque vous insérez ou supprimez des éléments dans une table, les index secondaires globaux sont également mis à jour. Ces mises à jour d'index consomment des unités de capacité d'écriture à partir de l'index, et non à partir de la table de base.

Par exemple, si vous exécutez une opération `Query` sur un index secondaire global et dépassez sa capacité de lecture approvisionnée, votre demande est limitée. Si vous exécutez une activité d'écriture massive sur la table, alors qu'un index secondaire global sur celle-ci a une capacité d'écriture insuffisante, l'activité d'écriture sur la table est limitée.

> **Important**
>
> Pour éviter les limitations potentielles, la capacité d'écriture allouée pour un index secondaire global doit être supérieure ou égale à la capacité d'écriture de la table de base, car les nouvelles mises à jour écrivent dans la table de base et l'index secondaire global.

Afin de modifier les paramètres de débit approvisionné pour un index secondaire global, utilisez l'opération `DescribeTable`. Des informations détaillées sur tous les index secondaires globaux de la table sont retournées.

Unités de capacité de lecture

Les index secondaires globaux prennent en charge les lectures éventuellement cohérentes ; chacune d'elles utilise la moitié d'une unité de capacité en lecture. Cela signifie qu'une interrogation d'index secondaire global peut extraire jusqu'à 2 x 4 Ko = 8 Ko par unité de capacité de lecture.

Pour les interrogations d'index secondaire global, DynamoDB calcule l'activité de lecture approvisionnée de la même façon que pour les interrogations de tables. La seule différence est que le calcul est basé sur les tailles des entrées d'index, plutôt que sur la taille de l'élément de la table de base. Le nombre d'unités de capacité en lecture est la somme des tailles de tous les attributs projetés de tous les éléments renvoyés. Le résultat est ensuite arrondi à la limite de 4 Ko suivante. Pour plus d'informations sur la façon dont DynamoDB calcule l'utilisation du débit approvisionné, consultez Mode de capacité provisionnée DynamoDB.

La taille maximum des résultats renvoyés par une opération `Query` est de 1 Mo. Cela inclut les tailles de tous les noms et valeurs d'attribut à travers l'ensemble des éléments retournés.

Par exemple, considérons un index secondaire global où chaque élément contient 2 000 octets de données. Supposons à présent que vous effectuez une opération `Query` sur cet index et que la `KeyConditionExpression` de la requête renvoie huit éléments. La taille totale des éléments correspondants est de 2 000 octets x 8 éléments = 16 000 octets. Ce résultat est ensuite arrondi à la limite de 4 Ko la plus proche. Les interrogations d'index secondaire global étant éventuellement cohérentes, le coût total est de 0,5 x (16 Ko / 4 Ko), soit 2 unités de capacité de lecture.

Unités de capacité d'écriture

Lors de l'ajout, de la mise à jour ou de la suppression d'un élément dans une table, si l'opération affecte un index secondaire global, celui-ci consomme les unités de capacité d'écriture approvisionnée à cette fin. Le coût total du débit alloué pour une écriture se compose de la somme des unités de capacité en écriture utilisées par l'écriture sur la table de base et de celles utilisées par la mise à jour des index secondaires globaux. Notez que, si une écriture dans une table ne requiert de mise à jour d'index secondaire global, aucune capacité d'écriture n'est consommée à partir de l'index.

Pour qu'une écriture de table réussisse, les paramètres de débit provisionnés de la table et de tous ses index secondaires globaux doivent avoir une capacité d'écriture suffisante pour accueillir l'écriture. Sinon, l'écriture dans la table est limitée.

> **Important**
>
> Lors de la création d'un index secondaire global (GSI), les opérations d'écriture dans la table de base peuvent être limitées si l'activité GSI résultant des écritures dans la table de base dépasse la capacité d'écriture allouée au GSI. Cette limitation affecte toutes les opérations d'écriture, du processus d'indexation à la perturbation potentielle de vos charges de travail de production. Pour plus d'informations, consultez Résolution des problèmes de limitation dans Amazon DynamoDB.

Le coût d'écriture d'un élément dans un index secondaire global dépend de plusieurs facteurs :

- Si vous écrivez un nouvel élément sur la table qui définit un attribut indexé, ou que vous mettez à jour un élément existant pour définir un attribut indexé précédemment non défini, une opération d'écriture est nécessaire pour insérer l'élément dans l'index.
- Si une mise à jour de la table change la valeur d'un attribut de clé indexée (de A en B), deux écritures sont requises, une pour supprimer l'élément précédent de l'index et une autre pour insérer le nouveau élément dans l'index.

- Si un élément était présent dans l'index, mais qu'une écriture sur la table a entraîné la suppression de l'attribut indexé, une écriture est nécessaire pour supprimer de l'index la projection de l'ancien élément.
- Si un élément n'est pas présent dans l'index avant ou après la mise à jour de l'élément, il n'y a aucun coût d'écriture supplémentaire pour l'index.
- Si la mise à jour de la table ne modifie que la valeur des attributs projetés du schéma de clé d'index, mais ne change pas la valeur d'un quelconque attribut de clé indexé, une écriture est nécessaire pour mettre à jour les valeurs des attributs projetés sur l'index.

Tous ces facteurs partent du principe que la taille de chaque élément dans l'index est inférieure ou égale à la taille d'élément de 1 Ko pour le calcul des unités de capacité d'écriture. Les plus grandes entrées d'index nécessitent des unités de capacité en écriture supplémentaires. Vous pouvez réduire vos coûts d'écriture en considérant les attributs que vos requêtes ont besoin de retourner et en projetant uniquement ces attributs dans l'index.

Considérations relatives au stockage pour les index secondaires globaux

Quand une application écrit un élément dans une table, DynamoDB copie automatiquement le sous-ensemble approprié d'attributs vers les index secondaires globaux où ces attributs doivent apparaître. Votre AWS compte est débité pour le stockage de l'article dans la table de base ainsi que pour le stockage des attributs dans les index secondaires globaux de cette table.

La quantité d'espace utilisée par un élément de l'index est la somme des éléments suivants :

- La taille en octets de la clé primaire de la table de base (clé de partition et clé de tri)
- La taille en octets de l'attribut de clé d'index
- La taille en octets des attributs projetés (le cas échéant)
- 100 octets de surcharge par élément d'index

Afin d'estimer les exigences de stockage pour un index secondaire global, vous pouvez évaluer la taille moyenne d'un élément de l'index, puis la multiplier par le nombre d'éléments de la table de base ayant les attributs de clé d'index secondaire global.

Si une table contient un élément dont aucun attribut particulier n'est défini, mais que cet attribut est défini comme clé de partition d'index ou clé de tri, DynamoDB n'écrit aucune donnée pour cet élément dans l'index.

Motifs de design

Les modèles de conception fournissent des solutions éprouvées aux défis courants liés à l'utilisation d'index secondaires globaux. Ces modèles vous aident à créer des applications efficaces et évolutives en vous montrant comment structurer vos index pour des cas d'utilisation spécifiques.

Chaque modèle inclut un guide d'implémentation complet avec des exemples de code, les meilleures pratiques et des cas d'utilisation réels pour vous aider à appliquer le modèle à vos propres applications.

Rubriques

- Modèle de clés à attributs multiples

Modèle de clés à attributs multiples

Aperçu

Les clés à attributs multiples vous permettent de créer une partition d'index secondaire global (GSI) et de trier les clés composées d'un maximum de quatre attributs chacune. Cela réduit le code côté client et facilite la modélisation initiale des données et l'ajout de nouveaux modèles d'accès ultérieurement.

Imaginons un scénario courant : pour créer un GSI qui interroge des éléments selon plusieurs attributs hiérarchiques, vous devez traditionnellement créer des clés synthétiques en concaténant des valeurs. Par exemple, dans une application de jeu, pour interroger les matchs d'un tournoi par tournoi, région et manche, vous pouvez créer une clé de partition GSI synthétique telle que TOURNAMENT# WINTER2 024 #REGION #NA -EAST et une clé de tri synthétique telle que ROUND #SEMIFINALS #BRACKET #UPPER. Cette approche fonctionne, mais nécessite la concaténation de chaînes lors de l'écriture de données, l'analyse lors de la lecture et le remplissage de clés synthétiques sur tous les éléments existants si vous ajoutez le GSI à une table existante. Cela rend le code plus encombré et il est difficile de maintenir la sécurité des types sur les composants clés individuels.

Les clés à attributs multiples résolvent ce problème pour GSIs. Vous définissez votre clé de partition GSI à l'aide de plusieurs attributs existants tels que TournamentID et region. DynamoDB gère automatiquement la logique des clés composites, en les hachant ensemble pour la distribution des données. Vous écrivez des éléments en utilisant les attributs naturels de votre modèle de domaine, et le GSI les indexe automatiquement. Pas de concaténation, pas d'analyse syntaxique, pas de remblayage. Votre code reste propre, vos données restent saisies et vos requêtes restent simples.

Cette approche est particulièrement utile lorsque vous disposez de données hiérarchiques avec des groupements d'attributs naturels (par exemple tournoi → région → manche, ou organisation → département → équipe).

Exemple d'application

Ce guide explique comment créer un système de suivi des matchs de tournoi pour une plateforme d'e-sport. La plateforme doit interroger efficacement les matchs selon plusieurs critères : par tournoi et par région pour la gestion des brackets, par joueur pour l'historique des matchs et par date pour la programmation.

Modèle de données

Dans cette présentation, le système de suivi des matchs de tournoi prend en charge trois modèles d'accès principaux, chacun nécessitant une structure de clé différente :

Modèle d'accès 1 : recherchez une correspondance spécifique à l'aide de son identifiant unique

- Solution : table de base avec `matchId` comme clé de partition

Modèle d'accès 2 : recherchez tous les matchs pour un tournoi et une région spécifiques, en filtrant éventuellement par tour, bracket ou match

- Solution : index secondaire global avec clé de partition multi-attributs (`tournamentId+region`) et clé de tri multi-attributs (`round+ +bracket`) `matchId`
- Exemples de requêtes : « Tous les WINTER2 024 matchs dans la région NA-EAST » ou « Tous les matchs des DEMI-FINALES dans la tranche SUPÉRIEURE pour 024/NA-EAST » WINTER2

Modèle d'accès 3 : recherchez l'historique des matchs d'un joueur, en filtrant éventuellement par plage de dates ou par tour de tournoi

- Solution : index secondaire global avec clé de partition unique (`player1Id`) et clé de tri multi-attributs (`matchDate+round`)
- Exemples de requêtes : « Tous les matchs du joueur 101 » ou « Les matchs du joueur 101 en janvier 2024 »

La principale différence entre les approches traditionnelles et les approches à attributs multiples apparaît clairement lorsque l'on examine la structure des articles :

Approche traditionnelle de l'index secondaire global (clés concaténées) :

```
// Manual concatenation required for GSI keys
const item = {
    matchId: 'match-001',                                    // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    // Synthetic keys needed for GSI
    GSI_PK: `TOURNAMENT#${tournamentId}#REGION#${region}`,       // Must concatenate
    GSI_SK: `${round}#${bracket}#${matchId}`,                    // Must concatenate
    // ... other attributes
};
```

Approche de l'index secondaire global à attributs multiples (clés natives) :

```
// Use existing attributes directly - no concatenation needed
const item = {
    matchId: 'match-001',                                    // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    matchDate: '2024-01-18',
    // No synthetic keys needed - GSI uses existing attributes directly
    // ... other attributes
};
```

Avec les clés à attributs multiples, vous pouvez écrire des éléments une seule fois avec des attributs de domaine naturels. DynamoDB les indexe automatiquement sur GSIs plusieurs sans nécessiter de clés concaténées synthétiques.

Schéma de table de base :

- Clé de partition : `matchId` (1 attribut)

Schéma d'index secondaire global (TournamentRegionIndex avec clés à attributs multiples) :

- Clé de partition :`tournamentId`, `region` (2 attributs)

- Clé de tri : `roundbracket`,, `matchId` (3 attributs)

Schéma d'index secondaire global (PlayerMatchHistoryIndex avec clés à attributs multiples) :

- Clé de partition : `player1Id` (1 attribut)
- Clé de tri :`matchDate`, `round` (2 attributs)

Table de base : TournamentMatches

MatchID (PK)	ID du tournoi	region	round	crochet	ID du joueur 1	ID du joueur 2	Date du match	vainqueu	score
match-0(	WINTER 4	NA-EAST	FINALES	CHAMPI AT	101	103	20-01-20 4	101	3-1
match-0(	WINTER 4	NA-EAST	DEMI-FINA LES	UPPER	101	105	18-01-20 4	101	3-2
match-0(	WINTER 4	NA-EAST	DEMI-FINA LES	UPPER	103	107	18-01-20 4	103	3-0
match-0(	WINTER 4	NA-EAST	QUARTS DE FINALE	UPPER	101	109	15-01-20 4	101	3-1
match-0(	WINTER 4	NA-WEST	FINALES	CHAMPI AT	102	104	20-01-20 4	102	3-2
match-0(	WINTER 4	NA-WEST	DEMI-FINA LES	UPPER	102	106	18-01-20 4	102	3-1
match-0(	SPRING 4	NA-EAST	QUARTS DE FINALE	UPPER	101	108	15/03/20 4	101	3-0

MatchID (PK)	ID du tournoi	region	round	crochet	ID du joueur 1	ID du joueur 2	Date du match	vainqueu	score
match-0(	SPRING 4	NA-EAST	QUARTS DE FINALE	LOWER	103	110	15/03/20 4	103	3-2

GSI : TournamentRegionIndex (clés à attributs multiples)

ID du tournoi (PK)	région (PK)	rond (SK)	support (SK)	MatchID (SK)	ID du joueur 1	ID du joueur 2	Date du match	vainqueu	score
WINTER 4	NA-EAST	FINALES	CHAMPI AT	match-0(	101	103	20-01-20 4	101	3-1
WINTER 4	NA-EAST	QUARTS DE FINALE	UPPER	match-0(	101	109	15-01-20 4	101	3-1
WINTER 4	NA-EAST	DEMI-FINA LES	UPPER	match-0(	101	105	18-01-20 4	101	3-2
WINTER 4	NA-EAST	DEMI-FINA LES	UPPER	match-0(	103	107	18-01-20 4	103	3-0
WINTER 4	NA-WEST	FINALES	CHAMPI AT	match-0(	102	104	20-01-20 4	102	3-2
WINTER 4	NA-WEST	DEMI-FINA LES	UPPER	match-0(	102	106	18-01-20 4	102	3-1

ID du tournoi (PK)	région (PK)	rond (SK)	support (SK)	MatchID (SK)	ID du joueur 1	ID du joueur 2	Date du match	vainqueu	score
SPRING 4	NA-EAST	QUARTS DE FINALE	LOWER	match-0(	103	110	15/03/20 4	103	3-2
SPRING 4	NA-EAST	QUARTS DE FINALE	UPPER	match-0(	101	108	15/03/20 4	101	3-0

GSI : PlayerMatchHistoryIndex (clés à attributs multiples)

ID du joueur 1 (PK)	MatchDa (SK)	rond (SK)	ID du tournoi	region	crochet	Identifia nt du match	ID du joueur 2	vainqueu	score
101	15-01-20 4	QUARTS DE FINALE	WINTER 4	NA-EAST	UPPER	match-0(	109	101	3-1
101	18-01-20 4	DEMI-FINALES	WINTER 4	NA-EAST	UPPER	match-0(	105	101	3-2
101	20-01-20 4	FINALES	WINTER 4	NA-EAST	CHAMPI AT	match-0(	103	101	3-1
101	15/03/20 4	QUARTS DE FINALE	SPRING 4	NA-EAST	UPPER	match-0(	108	101	3-0
102	18-01-20 4	DEMI-FINALES	WINTER 4	NA-WEST	UPPER	match-0(	106	102	3-1

ID du joueur 1 (PK)	MatchDa (SK)	rond (SK)	ID du tournoi	region	crochet	Identifiant du match	ID du joueur 2	vainqueu	score
102	20-01-20 4	FINALES	WINTER 4	NA-WEST	CHAMPI AT	match-0(	104	102	3-2
103	18-01-20 4	DEMI-FINA LES	WINTER 4	NA-EAST	UPPER	match-0(	107	103	3-0
103	15/03/20 4	QUARTS DE FINALE	SPRING 4	NA-EAST	LOWER	match-0(	110	103	3-2

Prérequis

Avant de commencer, assurez-vous de disposer des éléments suivants :

Compte et autorisations

- Un AWS compte actif (créez-en un ici si nécessaire)
- Autorisations IAM pour les opérations DynamoDB :
  - `dynamodb:CreateTable`
  - `dynamodb:DeleteTable`
  - `dynamodb:DescribeTable`
  - `dynamodb:PutItem`
  - `dynamodb:Query`
  - `dynamodb:BatchWriteItem`

> **Note**
>
> Note de sécurité : Pour une utilisation en production, créez une politique IAM personnalisée avec uniquement les autorisations dont vous avez besoin. Pour ce didacticiel, vous pouvez utiliser la politique AWS géréeAmazonDynamoDBFullAccessV2.

Environnement de développement

- Node.js installé sur votre machine
- AWS informations d'identification configurées à l'aide de l'une des méthodes suivantes :

Option 1 : AWS CLI

```
aws configure
```

Option 2 : variables d’environnement

```
export AWS_ACCESS_KEY_ID=your_access_key_here
export AWS_SECRET_ACCESS_KEY=your_secret_key_here
export AWS_DEFAULT_REGION=us-east-1
```

Installation des packages obligatoires

```
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
```

Mise en œuvre

Étape 1 : Création d'une table à GSIs l'aide de clés à attributs multiples

Créez une table avec une structure de clés de base simple et utilisant GSIs des clés à attributs multiples.

Exemple de code

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });

const response = await client.send(new CreateTableCommand({
    TableName: 'TournamentMatches',

    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'matchId', KeyType: 'HASH' }             // Simple PK
    ],

    AttributeDefinitions: [
```

```
        { AttributeName: 'matchId', AttributeType: 'S' },
        { AttributeName: 'tournamentId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'round', AttributeType: 'S' },
        { AttributeName: 'bracket', AttributeType: 'S' },
        { AttributeName: 'player1Id', AttributeType: 'S' },
        { AttributeName: 'matchDate', AttributeType: 'S' }
    ],

    // GSIs with multi-attribute keys
    GlobalSecondaryIndexes: [
        {
            IndexName: 'TournamentRegionIndex',
            KeySchema: [
                { AttributeName: 'tournamentId', KeyType: 'HASH' },    // GSI PK
 attribute 1
                { AttributeName: 'region', KeyType: 'HASH' },          // GSI PK
 attribute 2
                { AttributeName: 'round', KeyType: 'RANGE' },          // GSI SK
 attribute 1
                { AttributeName: 'bracket', KeyType: 'RANGE' },        // GSI SK
 attribute 2
                { AttributeName: 'matchId', KeyType: 'RANGE' }         // GSI SK
 attribute 3
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'PlayerMatchHistoryIndex',
            KeySchema: [
                { AttributeName: 'player1Id', KeyType: 'HASH' },       // GSI PK
                { AttributeName: 'matchDate', KeyType: 'RANGE' },      // GSI SK
 attribute 1
                { AttributeName: 'round', KeyType: 'RANGE' }           // GSI SK
 attribute 2
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],

    BillingMode: 'PAY_PER_REQUEST'
}));
```

```
console.log("Table with multi-attribute GSI keys created successfully");
```

Principales décisions de conception :

Table de base : La table de base utilise une clé de `matchId` partition simple pour les recherches par correspondance directe, ce qui permet de maintenir la structure de la table de base simple tout en GSIs fournissant des modèles de requête complexes.

TournamentRegionIndex Index secondaire global : L'index secondaire `TournamentRegionIndex` mondial utilise `tournamentId` + `region` comme clé de partition à attributs multiples, ce qui crée une isolation entre les régions de tournois, les données étant distribuées par le hachage des deux attributs combinés, ce qui permet d'effectuer des requêtes efficaces dans un contexte de région de tournoi spécifique. La clé de tri à attributs multiples (`round`+ `bracket` +`matchId`) permet un tri hiérarchique qui prend en charge les requêtes à tous les niveaux de la hiérarchie avec un ordre naturel, du général (rond) au spécifique (identifiant de correspondance).

PlayerMatchHistoryIndex Index secondaire mondial : L'index secondaire `PlayerMatchHistoryIndex` mondial réorganise les données par joueur en les utilisant `player1Id` comme clé de partition, ce qui permet d'effectuer des requêtes entre tournois pour un joueur spécifique. La clé de tri à attributs multiples (`matchDate+round`) fournit un ordre chronologique avec la possibilité de filtrer par plages de dates ou par tours de tournoi spécifiques.

Étape 2 : Insérer des données avec des attributs natifs

Ajoutez les données des matchs du tournoi à l'aide d'attributs naturels. Le GSI indexera automatiquement ces attributs sans nécessiter de clés synthétiques.

Exemple de code

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Tournament match data - no synthetic keys needed for GSIs
const matches = [
    // Winter 2024 Tournament, NA-EAST region
    {
        matchId: 'match-001',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
```

```
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '101',
        player2Id: '103',
        matchDate: '2024-01-20',
        winner: '101',
        score: '3-1'
    },
    {
        matchId: 'match-002',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '105',
        matchDate: '2024-01-18',
        winner: '101',
        score: '3-2'
    },
    {
        matchId: 'match-003',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '103',
        player2Id: '107',
        matchDate: '2024-01-18',
        winner: '103',
        score: '3-0'
    },
    {
        matchId: 'match-004',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '109',
        matchDate: '2024-01-15',
        winner: '101',
        score: '3-1'
    },
```

```
    // Winter 2024 Tournament, NA-WEST region
    {
        matchId: 'match-005',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '102',
        player2Id: '104',
        matchDate: '2024-01-20',
        winner: '102',
        score: '3-2'
    },
    {
        matchId: 'match-006',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '102',
        player2Id: '106',
        matchDate: '2024-01-18',
        winner: '102',
        score: '3-1'
    },

    // Spring 2024 Tournament, NA-EAST region
    {
        matchId: 'match-007',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '108',
        matchDate: '2024-03-15',
        winner: '101',
        score: '3-0'
    },
    {
        matchId: 'match-008',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
```

```
        round: 'QUARTERFINALS',
        bracket: 'LOWER',
        player1Id: '103',
        player2Id: '110',
        matchDate: '2024-03-15',
        winner: '103',
        score: '3-2'
    }
];

// Insert all matches
for (const match of matches) {
    await docClient.send(new PutCommand({
        TableName: 'TournamentMatches',
        Item: match
    }));

    console.log(`Added: ${match.matchId} - ${match.tournamentId}/${match.region} -
 ${match.round} ${match.bracket}`);
}

console.log(`\nInserted ${matches.length} tournament matches`);
console.log("No synthetic keys created - GSIs use native attributes automatically");
```

Structure des données expliquée :

Utilisation naturelle des attributs : chaque attribut représente un véritable concept de tournoi sans aucune concaténation ou analyse de chaînes requise, fournissant ainsi un mappage direct avec le modèle de domaine.

Indexation automatique de l'index secondaire global : Indexation GSIs automatique des éléments à l'aide des attributs existants (`tournamentIdregion`,`round`,`bracket`, `matchId` pour TournamentRegionIndex et `player1IdmatchDate`, `round` pour PlayerMatchHistoryIndex) sans nécessiter de clés concaténées synthétiques.

Aucun remplissage supplémentaire n'est nécessaire : lorsque vous ajoutez un nouvel index secondaire global avec des clés à attributs multiples à une table existante, DynamoDB indexe automatiquement tous les éléments existants à l'aide de leurs attributs naturels. Il n'est pas nécessaire de mettre à jour les éléments à l'aide de clés synthétiques.

Étape 3 : Interrogez TournamentRegionIndex l'index secondaire global avec tous les attributs de clé de partition

Cet exemple interroge l'index secondaire TournamentRegionIndex global qui possède une clé de partition à attributs multiples (`tournamentId+region`). Tous les attributs de clé de partition doivent être spécifiés avec des conditions d'égalité dans les requêtes. Vous ne pouvez pas effectuer de requête `tournamentId` uniquement ou utiliser des opérateurs d'inégalité sur les attributs de clé de partition.

Exemple de code

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query GSI: All matches for WINTER2024 tournament in NA-EAST region
const response = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
    ExpressionAttributeNames: {
        '#region': 'region',  // 'region' is a reserved keyword
        '#tournament': 'tournament'
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST'
    }
}));

console.log(`Found ${response.Items.length} matches for WINTER2024/NA-EAST:\n`);
response.Items.forEach(match => {
    console.log(`  ${match.round} | ${match.bracket} | ${match.matchId}`);
    console.log(`    Players: ${match.player1Id} vs ${match.player2Id}`);
    console.log(`    Winner: ${match.winner}, Score: ${match.score}\n`);
});
```

Résultat attendu :

```
Found 4 matches for WINTER2024/NA-EAST:
```

```
  FINALS | CHAMPIONSHIP | match-001
    Players: 101 vs 103
    Winner: 101, Score: 3-1

  QUARTERFINALS | UPPER | match-004
    Players: 101 vs 109
    Winner: 101, Score: 3-1

  SEMIFINALS | UPPER | match-002
    Players: 101 vs 105
    Winner: 101, Score: 3-2

  SEMIFINALS | UPPER | match-003
    Players: 103 vs 107
    Winner: 103, Score: 3-0
```

Requêtes non valides :

```
// Missing region attribute
KeyConditionExpression: 'tournamentId = :tournament'

// Using inequality on partition key attribute
KeyConditionExpression: 'tournamentId = :tournament AND #region > :region'
```

Performances : les clés de partition à attributs multiples sont hachées ensemble, offrant les mêmes performances de recherche O (1) que les clés à attribut unique.

Étape 4 : demander les clés de tri de l'index secondaire global left-to-right

Les attributs clés de tri doivent être interrogés left-to-right dans l'ordre dans lequel ils sont définis dans l'index secondaire global. Cet exemple montre comment interroger les différents TournamentRegionIndex niveaux hiérarchiques : filtrage par simple`round`, par `round` + `bracket` ou par les trois attributs clés de tri. Vous ne pouvez pas ignorer les attributs au milieu. Par exemple, vous ne pouvez pas effectuer d'interrogation par `round` et `matchId` pendant les sauts. `bracket`

Exemple de code

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
```

```
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Filter by first sort key attribute (round)
console.log("Query 1: All SEMIFINALS matches");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Filter by first two sort key attributes (round + bracket)
console.log("Query 2: SEMIFINALS UPPER bracket matches");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Filter by all three sort key attributes (round + bracket + matchId)
console.log("Query 3: Specific match in SEMIFINALS UPPER bracket");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
```

```
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket AND matchId = :matchId',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER',
        ':matchId': 'match-002'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - skipping round
console.log("Query 4: Attempting to skip first sort key attribute (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot skip sort key attributes - must query left-to-right\n`);
}
```

Résultat attendu :

```
Query 1: All SEMIFINALS matches
  Found 2 matches

Query 2: SEMIFINALS UPPER bracket matches
```

```
  Found 2 matches

Query 3: Specific match in SEMIFINALS UPPER bracket
  Found 1 matches

Query 4: Attempting to skip first sort key attribute (WILL FAIL)
  Error: Query key condition not supported
  Cannot skip sort key attributes - must query left-to-right
```

Left-to-right règles de requête : vous devez interroger les attributs dans l'ordre de gauche à droite, sans en sauter aucun.

Modèles valides :

- Premier attribut uniquement : `round = 'SEMIFINALS'`
- Les deux premiers attributs : `round = 'SEMIFINALS' AND bracket = 'UPPER'`
- Les trois attributs : `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId = 'match-002'`

Modèles non valides :

- Ignorer le premier attribut : `bracket = 'UPPER'` (saute le tour)
- Interrogation hors ordre : `matchId = 'match-002' AND round = 'SEMIFINALS'`
- Laisser des espaces : `round = 'SEMIFINALS' AND matchId = 'match-002'` (ne tient pas compte du crochet)

> **Note**
>
> Conseil de conception : Triez les attributs clés du plus général au plus spécifique afin d'optimiser la flexibilité des requêtes.

Étape 5 : Utiliser les conditions d'inégalité sur les clés de tri de l'indice secondaire global

Les conditions d'inégalité doivent être la dernière condition de votre requête. Cet exemple montre comment utiliser les opérateurs de comparaison (`>=`,`BETWEEN`) et la correspondance de préfixes (`begins_with()`) sur les attributs clés de tri. Une fois que vous avez utilisé un opérateur d'inégalité,

vous ne pouvez pas ajouter de conditions clés de tri supplémentaires après celui-ci. L'inégalité doit être la condition finale de votre expression de condition clé.

Exemple de code

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Round comparison (inequality on first sort key attribute)
console.log("Query 1: Matches from QUARTERFINALS onwards");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 >= :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'QUARTERFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Round range with BETWEEN
console.log("Query 2: Matches between QUARTERFINALS and SEMIFINALS");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 BETWEEN :start AND :end',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':start': 'QUARTERFINALS',
        ':end': 'SEMIFINALS'
```

```
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Prefix matching with begins_with (treated as inequality)
console.log("Query 3: Matches in brackets starting with 'U'");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND begins_with(bracket, :prefix)',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':prefix': 'U'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - condition after inequality
console.log("Query 4: Attempting condition after inequality (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 round > :round AND bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':round': 'QUARTERFINALS',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot add conditions after inequality - it must be last\n`);
```

```
}
```

Règles relatives aux opérateurs d'inégalité : vous pouvez utiliser des opérateurs de comparaison (>>=,<,,<=) BETWEEN pour les requêtes de plage et `begins_with()` pour la correspondance de préfixes. L'inégalité doit être la dernière condition de votre requête.

Modèles valides :

- Conditions d'égalité suivies d'inégalités : `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId > 'match-001'`
- Inégalité sur le premier attribut : `round BETWEEN 'QUARTERFINALS' AND 'SEMIFINALS'`
- Correspondance du préfixe comme condition finale : `round = 'SEMIFINALS' AND begins_with(bracket, 'U')`

Modèles non valides :

- Ajouter des conditions après une inégalité : `round > 'QUARTERFINALS' AND bracket = 'UPPER'`
- En utilisant de multiples inégalités : `round > 'QUARTERFINALS' AND bracket > 'L'`

> **Important**
>
> `begins_with()`est traitée comme une condition d'inégalité, de sorte qu'aucune condition clé de tri supplémentaire ne peut la suivre.

Étape 6 : Interroger l'index secondaire PlayerMatchHistoryIndex global avec une clé de tri à attributs multiples

Cet exemple interroge celui PlayerMatchHistoryIndex qui possède une clé de partition unique (`player1Id`) et une clé de tri multi-attributs (`matchDate+round`). Cela permet d'effectuer une analyse entre tournois en interrogeant tous les matchs pour un joueur spécifique sans connaître le tournoi, IDs alors que le tableau de base nécessiterait des requêtes distinctes par combinaison tournien-région.

Exemple de code

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: All matches for Player 101 across all tournaments
console.log("Query 1: All matches for Player 101");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player',
    ExpressionAttributeValues: {
        ':player': '101'
    }
}));

console.log(`  Found ${query1.Items.length} matches for Player 101:`);
query1.Items.forEach(match => {
    console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
});
console.log();

// Query 2: Player 101 matches on specific date
console.log("Query 2: Player 101 matches on 2024-01-18");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18'
    }
}));

console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Player 101 SEMIFINALS matches on specific date
console.log("Query 3: Player 101 SEMIFINALS matches on 2024-01-18");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date AND round
 = :round',
```

```
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18',
        ':round': 'SEMIFINALS'
    }
}));

console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: Player 101 matches in date range
console.log("Query 4: Player 101 matches in January 2024");
const query4 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate BETWEEN :start
 AND :end',
    ExpressionAttributeValues: {
        ':player': '101',
        ':start': '2024-01-01',
        ':end': '2024-01-31'
    }
}));

console.log(`  Found ${query4.Items.length} matches\n`);
```

Variations de motifs

Données chronologiques avec clés à attributs multiples

Optimisation pour les requêtes de séries chronologiques avec des attributs temporels hiérarchiques

Exemple de code

```
{
    TableName: 'IoTReadings',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'readingId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'readingId', AttributeType: 'S' },
        { AttributeName: 'deviceId', AttributeType: 'S' },
        { AttributeName: 'locationId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
```

```
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'timestamp', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for time-series queries
    GlobalSecondaryIndexes: [{
        IndexName: 'DeviceLocationTimeIndex',
        KeySchema: [
            { AttributeName: 'deviceId', KeyType: 'HASH' },
            { AttributeName: 'locationId', KeyType: 'HASH' },
            { AttributeName: 'year', KeyType: 'RANGE' },
            { AttributeName: 'month', KeyType: 'RANGE' },
            { AttributeName: 'day', KeyType: 'RANGE' },
            { AttributeName: 'timestamp', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query patterns enabled via GSI:
// - All readings for device in location
// - Readings for specific year
// - Readings for specific month in year
// - Readings for specific day
// - Readings in time range
```

Avantages : La hiérarchie temporelle naturelle (année → mois → jour → horodatage) permet des requêtes efficaces à tout moment avec une granularité sans analyse ou manipulation de dates. L'index secondaire global indexe automatiquement toutes les lectures en utilisant leurs attributs temporels naturels.

Commandes de commerce électronique avec clés à attributs multiples

Suivez les commandes avec plusieurs dimensions

Exemple de code

```
{
    TableName: 'Orders',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'orderId', KeyType: 'HASH' }
```

```
    ],
    AttributeDefinitions: [
        { AttributeName: 'orderId', AttributeType: 'S' },
        { AttributeName: 'sellerId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'orderDate', AttributeType: 'S' },
        { AttributeName: 'category', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'orderStatus', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'SellerRegionIndex',
            KeySchema: [
                { AttributeName: 'sellerId', KeyType: 'HASH' },
                { AttributeName: 'region', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'category', KeyType: 'RANGE' },
                { AttributeName: 'orderId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'CustomerOrdersIndex',
            KeySchema: [
                { AttributeName: 'customerId', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'orderStatus', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// SellerRegionIndex GSI queries:
// - Orders by seller and region
// - Orders by seller, region, and date
// - Orders by seller, region, date, and category

// CustomerOrdersIndex GSI queries:
// - Customer's orders
// - Customer's orders by date
```

```
// - Customer's orders by date and status
```

Données d'organisation hiérarchiques

Hiérarchies organisationnelles modélisées

Exemple de code

```
{
    TableName: 'Employees',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'employeeId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'employeeId', AttributeType: 'S' },
        { AttributeName: 'companyId', AttributeType: 'S' },
        { AttributeName: 'divisionId', AttributeType: 'S' },
        { AttributeName: 'departmentId', AttributeType: 'S' },
        { AttributeName: 'teamId', AttributeType: 'S' },
        { AttributeName: 'skillCategory', AttributeType: 'S' },
        { AttributeName: 'skillLevel', AttributeType: 'S' },
        { AttributeName: 'yearsExperience', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'OrganizationIndex',
            KeySchema: [
                { AttributeName: 'companyId', KeyType: 'HASH' },
                { AttributeName: 'divisionId', KeyType: 'HASH' },
                { AttributeName: 'departmentId', KeyType: 'RANGE' },
                { AttributeName: 'teamId', KeyType: 'RANGE' },
                { AttributeName: 'employeeId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'SkillsIndex',
            KeySchema: [
                { AttributeName: 'skillCategory', KeyType: 'HASH' },
                { AttributeName: 'skillLevel', KeyType: 'RANGE' },
                { AttributeName: 'yearsExperience', KeyType: 'RANGE' }
            ],
```

```
            Projection: { ProjectionType: 'INCLUDE', NonKeyAttributes: ['employeeId',
 'name'] }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// OrganizationIndex GSI query patterns:
// - All employees in company/division
// - Employees in specific department
// - Employees in specific team

// SkillsIndex GSI query patterns:
// - Employees by skill and experience level
```

Clés à attributs multiples éparses

Combinez des clés à attributs multiples pour créer un GSI clairsemé

Exemple de code

```
{
    TableName: 'Products',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'productId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'productId', AttributeType: 'S' },
        { AttributeName: 'categoryId', AttributeType: 'S' },
        { AttributeName: 'subcategoryId', AttributeType: 'S' },
        { AttributeName: 'averageRating', AttributeType: 'N' },
        { AttributeName: 'reviewCount', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'CategoryIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'subcategoryId', KeyType: 'HASH' },
                { AttributeName: 'productId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
```

```
        {
            IndexName: 'ReviewedProductsIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'averageRating', KeyType: 'RANGE' },  // Optional
 attribute
                { AttributeName: 'reviewCount', KeyType: 'RANGE' }     // Optional
 attribute
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Only products with reviews appear in ReviewedProductsIndex GSI
// Automatic filtering without application logic
// Multi-attribute sort key enables rating and count queries
```

SaaS multi-tenant

Plateforme SaaS multi-locataires avec isolation du client

Exemple de code

```
// Table design
{
    TableName: 'SaasData',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'resourceId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'resourceId', AttributeType: 'S' },
        { AttributeName: 'tenantId', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'resourceType', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for tenant-customer isolation
    GlobalSecondaryIndexes: [{
        IndexName: 'TenantCustomerIndex',
        KeySchema: [
            { AttributeName: 'tenantId', KeyType: 'HASH' },
            { AttributeName: 'customerId', KeyType: 'HASH' },
```

```
            { AttributeName: 'resourceType', KeyType: 'RANGE' },
            { AttributeName: 'resourceId', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query GSI: All resources for tenant T001, customer C001
const resources = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001'
    }
}));

// Query GSI: Specific resource type for tenant/customer
const documents = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer AND
 resourceType = :type',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001',
        ':type': 'document'
    }
}));
```

Avantages : requêtes efficaces dans le contexte locataire-client et organisation naturelle des données.

Transactions financières

Le système bancaire suit les transactions du compte en utilisant GSIs

Exemple de code

```
// Table design
{
    TableName: 'BankTransactions',
```

```
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'transactionId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'transactionId', AttributeType: 'S' },
        { AttributeName: 'accountId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'transactionType', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'AccountTimeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' },
                { AttributeName: 'day', KeyType: 'RANGE' },
                { AttributeName: 'transactionId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'TransactionTypeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'transactionType', KeyType: 'RANGE' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query AccountTimeIndex GSI: All transactions for account in 2023
const yearTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year',
    ExpressionAttributeNames: { '#year': 'year' },
```

```
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023'
    }
}));

// Query AccountTimeIndex GSI: Transactions in specific month
const monthTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year AND #month
 = :month',
    ExpressionAttributeNames: { '#year': 'year', '#month': 'month' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023',
        ':month': '11'
    }
}));

// Query TransactionTypeIndex GSI: Deposits in 2023
const deposits = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'TransactionTypeIndex',
    KeyConditionExpression: 'accountId = :account AND transactionType = :type AND #year
 = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':type': 'deposit',
        ':year': '2023'
    }
}));
```

Exemple complet

L'exemple suivant illustre les clés à attributs multiples, de la configuration au nettoyage :

Exemple de code

```
import {
    DynamoDBClient,
    CreateTableCommand,
    DeleteTableCommand,
```

```
    waitUntilTableExists
} from "@aws-sdk/client-dynamodb";
import {
    DynamoDBDocumentClient,
    PutCommand,
    QueryCommand
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

async function multiAttributeKeysDemo() {
    console.log("Starting Multi-Attribute GSI Keys Demo\n");

    // Step 1: Create table with GSIs using multi-attribute keys
    console.log("1. Creating table with multi-attribute GSI keys...");
    await client.send(new CreateTableCommand({
        TableName: 'TournamentMatches',
        KeySchema: [
            { AttributeName: 'matchId', KeyType: 'HASH' }
        ],
        AttributeDefinitions: [
            { AttributeName: 'matchId', AttributeType: 'S' },
            { AttributeName: 'tournamentId', AttributeType: 'S' },
            { AttributeName: 'region', AttributeType: 'S' },
            { AttributeName: 'round', AttributeType: 'S' },
            { AttributeName: 'bracket', AttributeType: 'S' },
            { AttributeName: 'player1Id', AttributeType: 'S' },
            { AttributeName: 'matchDate', AttributeType: 'S' }
        ],
        GlobalSecondaryIndexes: [
            {
                IndexName: 'TournamentRegionIndex',
                KeySchema: [
                    { AttributeName: 'tournamentId', KeyType: 'HASH' },
                    { AttributeName: 'region', KeyType: 'HASH' },
                    { AttributeName: 'round', KeyType: 'RANGE' },
                    { AttributeName: 'bracket', KeyType: 'RANGE' },
                    { AttributeName: 'matchId', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            },
            {
                IndexName: 'PlayerMatchHistoryIndex',
```

```
                KeySchema: [
                    { AttributeName: 'player1Id', KeyType: 'HASH' },
                    { AttributeName: 'matchDate', KeyType: 'RANGE' },
                    { AttributeName: 'round', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            }
        ],
        BillingMode: 'PAY_PER_REQUEST'
    }));

    await waitUntilTableExists({ client, maxWaitTime: 120 }, { TableName:
'TournamentMatches' });
    console.log("Table created\n");

    // Step 2: Insert tournament matches
    console.log("2. Inserting tournament matches...");
    const matches = [
        { matchId: 'match-001', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '101', player2Id: '103', matchDate:
'2024-01-20', winner: '101', score: '3-1' },
        { matchId: 'match-002', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'SEMIFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '105', matchDate:
'2024-01-18', winner: '101', score: '3-2' },
        { matchId: 'match-003', tournamentId: 'WINTER2024', region: 'NA-WEST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '102', player2Id: '104', matchDate:
'2024-01-20', winner: '102', score: '3-2' },
        { matchId: 'match-004', tournamentId: 'SPRING2024', region: 'NA-EAST', round:
'QUARTERFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '108', matchDate:
'2024-03-15', winner: '101', score: '3-0' }
    ];

    for (const match of matches) {
        await docClient.send(new PutCommand({ TableName: 'TournamentMatches', Item:
match }));
    }
    console.log(`Inserted ${matches.length} tournament matches\n`);

    // Step 3: Query GSI with multi-attribute partition key
    console.log("3. Query TournamentRegionIndex GSI: WINTER2024/NA-EAST matches");
    const gsiQuery1 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
```

```
        ExpressionAttributeNames: { '#region': 'region' },
        ExpressionAttributeValues: { ':tournament': 'WINTER2024', ':region': 'NA-
EAST' }
    }));

    console.log(`  Found ${gsiQuery1.Items.length} matches:`);
    gsiQuery1.Items.forEach(match => {
        console.log(`    ${match.round} - ${match.bracket} - ${match.winner} won`);
    });

    // Step 4: Query GSI with multi-attribute sort key
    console.log("\n4. Query PlayerMatchHistoryIndex GSI: All matches for Player 101");
    const gsiQuery2 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'PlayerMatchHistoryIndex',
        KeyConditionExpression: 'player1Id = :player',
        ExpressionAttributeValues: { ':player': '101' }
    }));

    console.log(`  Found ${gsiQuery2.Items.length} matches for Player 101:`);
    gsiQuery2.Items.forEach(match => {
        console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
    });

    console.log("\nDemo complete");
    console.log("No synthetic keys needed - GSIs use native attributes automatically");
}

async function cleanup() {
    console.log("Deleting table...");
    await client.send(new DeleteTableCommand({ TableName: 'TournamentMatches' }));
    console.log("Table deleted");
}

// Run demo
multiAttributeKeysDemo().catch(console.error);

// Uncomment to cleanup:
// cleanup().catch(console.error);
```

Échafaudage de code minimal

Exemple de code

```
// 1. Create table with GSI using multi-attribute keys
await client.send(new CreateTableCommand({
    TableName: 'MyTable',
    KeySchema: [
        { AttributeName: 'id', KeyType: 'HASH' }        // Simple base table PK
    ],
    AttributeDefinitions: [
        { AttributeName: 'id', AttributeType: 'S' },
        { AttributeName: 'attr1', AttributeType: 'S' },
        { AttributeName: 'attr2', AttributeType: 'S' },
        { AttributeName: 'attr3', AttributeType: 'S' },
        { AttributeName: 'attr4', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [{
        IndexName: 'MyGSI',
        KeySchema: [
            { AttributeName: 'attr1', KeyType: 'HASH' },    // GSI PK attribute 1
            { AttributeName: 'attr2', KeyType: 'HASH' },    // GSI PK attribute 2
            { AttributeName: 'attr3', KeyType: 'RANGE' },   // GSI SK attribute 1
            { AttributeName: 'attr4', KeyType: 'RANGE' }    // GSI SK attribute 2
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}));

// 2. Insert items with native attributes (no concatenation needed for GSI)
await docClient.send(new PutCommand({
    TableName: 'MyTable',
    Item: {
        id: 'item-001',
        attr1: 'value1',
        attr2: 'value2',
        attr3: 'value3',
        attr4: 'value4',
        // ... other attributes
    }
}));

// 3. Query GSI with all partition key attributes
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
```

```
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2'
    }
}));

// 4. Query GSI with sort key attributes (left-to-right)
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2 AND attr3 = :v3',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2',
        ':v3': 'value3'
    }
}));

// Note: If any attribute name is a DynamoDB reserved keyword, use
 ExpressionAttributeNames:
// KeyConditionExpression: 'attr1 = :v1 AND #attr2 = :v2'
// ExpressionAttributeNames: { '#attr2': 'attr2' }
```

Ressources supplémentaires

- Meilleures pratiques DynamoDB
- Utilisation de tables et de données
- Index secondaires globaux
- Opérations de requête et de numérisation

Gestion des index secondaires globaux dans DynamoDB

Cette section décrit comment créer, modifier et supprimer des index secondaires globaux dans Amazon DynamoDB.

Rubriques

- Création d'une table avec des index secondaires globaux
- Description des index secondaires globaux sur une table

- Ajout d'un index secondaire global à une table existante
- Suppression d'un index secondaire global
- Modification d'un index secondaire global pendant la création

Création d'une table avec des index secondaires globaux

Pour créer une table avec un ou plusieurs index secondaires globaux, utilisez l'opération `CreateTable` avec le paramètre `GlobalSecondaryIndexes`. Pour bénéficier d'une flexibilité de requête maximum, vous pouvez créer jusqu'à 20 index secondaires globaux (quota par défaut) par table.

Vous devez spécifier un attribut faisant office de clé de partition d'index. Vous pouvez éventuellement spécifier un autre attribut pour la clé de tri d'index. Il n'est pas nécessaire que l'un de ces attributs de clé soit identique à un attribut de clé dans la table. Par exemple, dans le GameScorestableau (voirUtilisation d'index secondaires globaux dans DynamoDB), il n'`TopScoreDateTimey` a `TopScore` pas non plus d'attributs clés. Vous pouvez créer un index secondaire global avec une clé de partition `TopScore` et une clé de tri `TopScoreDateTime`. Vous pouvez utiliser un tel index pour déterminer s'il existe une corrélation entre les scores d'une partie et l'heure de la journée à laquelle celle-ci est jouée.

Chaque attribut de clé d'index doit être un scalaire de type `String`, `Number` ou `Binary` (il ne peut pas être de type document ou ensemble). Vous pouvez projeter des attributs de tout type de données dans un index secondaire global. Celui-ci inclut des scalaires, des documents et des ensembles. Pour obtenir la liste complète des types de données, consultez Types de données.

Si vous utilisez le mode approvisionné, vous devez fournir les paramètres `ProvisionedThroughput` pour l'index, à savoir `ReadCapacityUnits` et `WriteCapacityUnits`. Ces paramètres de débit approvisionné sont distincts de ceux de la table, mais se comportent de la même manière. Pour de plus amples informations, veuillez consulter Considérations sur le débit alloué pour les index secondaires globaux.

Les index secondaires globaux héritent du mode read/write capacité de la table de base. Pour de plus amples informations, veuillez consulter Considérations relatives au changement de mode de capacité dans DynamoDB.

> **Note**
>
> Lors de la création d'un nouvel index secondaire global, il peut être important de vérifier si la clé de partition de votre choix produit une distribution inégale ou réduite des données ou du

trafic entre les valeurs des clés de partition du nouvel index. Si cela se produit, vous pourriez voir des opérations de remplissage et d'écriture se produire en même temps et limiter les écritures dans la table de base. Le service prend des mesures pour minimiser l'éventualité de ce scénario, mais il n'a aucune idée de la forme des données client par rapport à la clé de partition d'index, à la projection choisie ou à la rareté de la clé primaire d'index.
Si vous pensez que votre nouvel index secondaire global peut présenter des données ou une distribution de trafic étroites ou asymétriques entre les valeurs des clés de partition, tenez compte des points suivants avant d'ajouter de nouveaux index à des tables importantes sur le plan opérationnel.

- Il peut être plus sûr d'ajouter l'index au moment où votre application génère le moins de trafic.
- Envisagez d'activer CloudWatch Contributor Insights sur votre table de base et vos index. Cela vous procurera des informations précieuses sur la distribution de votre trafic.
- Surveillez `WriteThrottleEvents` et `ThrottledRequests` mesurez `OnlineIndexPercentageProgress` CloudWatch tout au long du processus. Ajustez la capacité d'écriture allouée selon les besoins pour terminer le remblayage dans un délai raisonnable sans aucun effet de limitation significatif sur vos opérations en cours. `OnlineIndexConsumedWriteCapacity`et `OnlineThrottleEvents` devraient afficher 0 lors du remblayage de l'indice.
- Préparez-vous à annuler la création de l'index si vous subissez un impact opérationnel dû à la limitation des écritures.

Description des index secondaires globaux sur une table

Pour afficher l'état de tous les index secondaires globaux sur une table, utilisez l'opération `DescribeTable`. La portion `GlobalSecondaryIndexes` de la réponse affiche tous les index sur la table, ainsi que l'état actuel de chacun d'eux (`IndexStatus`).

L'état `IndexStatus` d'un index secondaire global peut être l'un des suivants :

- `CREATING` – L'index est en cours de création et n'est pas encore disponible pour utilisation.
- `ACTIVE` – L'index est prêt pour utilisation et les applications peuvent effectuer des opérations `Query` dessus.
- `UPDATING` – Les paramètres de débit approvisionné de l'index sont en cours de modification.
- `DELETING` – L'index est actuellement en cours de suppression et ne peut plus être utilisé.

Quand DynamoDB a fini de générer un index secondaire global, l'état de celui-ci passe de `CREATING` à `ACTIVE`.

Ajout d'un index secondaire global à une table existante

Pour ajouter un index secondaire global à une table existante, utilisez l'opération `UpdateTable` avec le paramètre `GlobalSecondaryIndexUpdates`. Vous devez fournir les informations suivantes :

- Un nom d'index. Le nom doit être unique parmi tous les index sur cette table.
- Le schéma de clé de l'index. Vous devez spécifier un attribut pour la clé de partition d'index. Vous pouvez éventuellement spécifier un autre attribut pour la clé de tri d'index. Il n'est pas nécessaire que l'un de ces attributs de clé soit identique à un attribut de clé dans la table. Les types de données pour chaque attribut de schéma doivent être scalaires : `String`, `Number` ou `Binary`.
- Attributs à projeter à partir de la table dans l'index :
  - `KEYS_ONLY` – Chaque élément de l'index se compose uniquement des valeurs de clé de partition et de tri de la table, ainsi que des valeurs de clé d'index.
  - `INCLUDE` – En plus des attributs décrits dans `KEYS_ONLY`, l'index secondaire inclut des attributs autres que de clé que vous spécifiez.
  - `ALL` – L'index inclut tous les attributs de la table source.
- Les paramètres de débit approvisionné pour l'index, à savoir `ReadCapacityUnits` et `WriteCapacityUnits`. Ces paramètres de débit approvisionné sont distincts de ceux de la table.

Vous ne pouvez créer qu'un seul index secondaire global par opération `UpdateTable`.

> **Important**
>
> Lors de la création d'un index secondaire global (GSI), les opérations d'écriture dans la table de base peuvent être limitées si l'activité GSI résultant des écritures dans la table de base dépasse la capacité d'écriture allouée au GSI. Cette limitation affecte toutes les opérations d'écriture, du processus d'indexation à la perturbation potentielle de vos charges de travail de production. Pour plus d'informations, consultez Résolution des problèmes de limitation dans Amazon DynamoDB.

Phases de création d'index

Lorsque vous ajoutez un nouvel index secondaire global à une table existante, la table reste disponible pendant la création de l'index. Toutefois, le nouvel index n'est pas disponible pour les opérations Query tant que son état n'est pas passé de `CREATING` à `ACTIVE`.

> **Note**
>
> La création d'un index secondaire global n'utilise pas la scalabilité automatique des applications. L'augmentation de la capacité de scalabilité automatique des applications `MIN` ne réduit pas le temps de création de l'index secondaire global.

En coulisses, DynamoDB génère l'index en deux phases :

Allocation de ressources

DynamoDB alloue les ressources de calcul et de stockage nécessaires à la génération de l'index.

Au cours de la phase d'allocation des ressources, l'attribut `IndexStatus` est `CREATING`, et l'attribut `Backfilling` a la valeur false. Utilisez l'opération `DescribeTable` pour récupérer l'état d'une table et de tous ses index secondaires.

Pendant que l'index est en phase d'allocation de ressources, vous ne pouvez pas supprimer l'index ou sa table parent. Vous ne pouvez pas non plus modifier le débit approvisionné de l'index ou de la table. Vous ne pouvez pas ajouter ou supprimer d'autres index sur la table. En revanche, vous pouvez modifier le débit approvisionné de ces autres index.

Remplissage

Pour chaque élément de la table, DynamoDB détermine l'ensemble d'attributs à écrire dans l'index en fonction de sa projection (`KEYS_ONLY`, `INCLUDE` ou `ALL`). Il écrit ensuite ces attributs dans l'index. Durant la phase de remplissage, DynamoDB suit les éléments ajoutés, supprimés ou mis à jour dans la table. Les attributs de ces éléments sont également ajoutés, supprimés ou mis à jour dans l'index, selon le cas.

Au cours de la phase de remplissage, l'attribut `IndexStatus` est défini sur `CREATING`, et l'attribut `Backfilling` a la valeur true. Utilisez l'opération `DescribeTable` pour récupérer l'état d'une table et de tous ses index secondaires.

Pendant le remplissage de l'index, vous ne pouvez pas supprimer sa table parent. Vous pouvez cependant toujours supprimer l'index ou modifier le débit approvisionné de la table et de n'importe lequel de ses index secondaires globaux.

> **Note**
>
> Pendant la phase de remplissage, certaines écritures d'éléments violant le schéma de clé peuvent aboutir tandis que d'autres sont rejetées. Après le remplissage, toutes les écritures dans les éléments, qui violent le schéma de clé du nouvel index sont rejetées. Nous vous recommandons d'exécuter l'outil de détection des violations une fois la phase de remplissage terminée afin de détecter et résoudre d'éventuelles violations de clé. Pour de plus amples informations, veuillez consulter Détection et correction des violations de clé d'index dans DynamoDB.

Pendant les phases d'allocation de ressources et de remplissage, l'index est dans l'état `CREATING`. Pendant ce temps, DynamoDB effectue des opérations de lecture sur la table. Vous n'êtes pas facturé pour les opérations de lecture de la table de base destinées au remplissage de l'index secondaire global.

Une fois la génération de l'index terminée, l'état passe à `ACTIVE`. Vous ne pouvez pas effectuer d'opération `Query` ou `Scan` sur l'index tant que l'état de celui-ci n'est pas `ACTIVE`.

> **Note**
>
> Dans certains cas, DynamoDB ne peut pas écrire des données de la table dans l'index en raison de violations de clé d'index. Cela peut se produire si :
>
> - Le type de données d'une valeur d'attribut ne correspond pas au type de données d'un type de données de schéma de clé d'index.
> - La taille d'un attribut dépasse la longueur maximale définie pour un attribut de clé d'index.
> - Un attribut de clé d'index a une valeur de String ou de Binary attribute vide.
>
> Les violations de clé d'index n'interfèrent pas avec la création d'index secondaire global. Cependant, quand l'index passe à l'état `ACTIVE`, il ne contient plus de clé violant le schéma de clé.

DynamoDB fournit un outil autonome pour détecter et résoudre ces problèmes. Pour de plus amples informations, veuillez consulter Détection et correction des violations de clé d'index dans DynamoDB.

Ajout d'un index secondaire global à une table volumineuse

Le temps nécessaire à la génération d'un index secondaire global dépend de plusieurs facteurs, tels que les suivants :

- La taille de la table
- Le nombre d'éléments dans la table pouvant être inclus dans l'index
- Le nombre d'attributs projetés dans l'index
- L'activité d'écriture sur la table principale pendant les générations d'index

Si vous ajoutez un index secondaire global à une table très volumineuse, le processus de création peut prendre beaucoup de temps. Pour suivre la progression et déterminer si la capacité d'écriture de l'index est suffisante, consultez les CloudWatch statistiques Amazon suivantes :

- `OnlineIndexPercentageProgress`

Pour plus d'informations sur CloudWatch les métriques associées à DynamoDB, consultez. Métriques DynamoDB

Important

Vous devrez peut-être autoriser la mise en liste d'autorisation de très grandes tables avant de créer ou de mettre à jour un index secondaire global. Contactez le AWS Support pour autoriser la mise en liste de vos tables.

Pendant le remplissage d'un index, DynamoDB utilise la capacité système interne pour lire la table. Cela permet de minimiser l'impact de la création d'index et de s'assurer que la table ne manque pas de capacité de lecture.

Suppression d'un index secondaire global

Si vous n'avez plus besoin d'un index secondaire global, vous pouvez le supprimer à l'aide de l'opération `UpdateTable`.

Vous ne pouvez supprimer qu'un seul index secondaire global par opération `UpdateTable`.

Le processus de suppression de l'index secondaire global n'a aucune incidence sur les activités de lecture ou d'écriture dans la table parent. Pendant la suppression, vous pouvez toujours modifier le débit approvisionné sur d'autres index.

> **Note**
>
> - Lorsque vous supprimez une table à l'aide de l'action `DeleteTable`, tous les index secondaires globaux sur cette table sont également supprimés.
> - Votre compte ne sera pas facturé pour l'opération de suppression de l'index secondaire global.

Modification d'un index secondaire global pendant la création

Pendant la génération d'un index, vous pouvez utiliser l'opération `DescribeTable` pour déterminer la phase en cours. La description de l'index inclut un attribut booléen, `Backfilling`, indiquant si DynamoDB est en train de charger l'index avec les éléments de la table. Si la valeur de `Backfilling` est true, la phase d'allocation de ressources est terminée et le remplissage de l'index est en cours.

Pendant la phase de remplissage, vous pouvez supprimer l'index en cours de création. Au cours de cette phase, vous ne pouvez pas ajouter ou supprimer d'autres index sur la table.

> **Note**
>
> Pour les index créés dans le cadre d'une opération `CreateTable`, l'attribut `Backfilling` n'apparaît pas dans la sortie `DescribeTable`. Pour de plus amples informations, veuillez consulter Phases de création d'index.

Détection et correction des violations de clé d'index dans DynamoDB

Durant la phase de remplissage du processus de création d'index secondaire global, Amazon DynamoDB examine chaque élément de la table pour déterminer s'il peut être inclus dans l'index. Il se peut que certains éléments ne soient pas être éligibles parce qu'ils provoqueraient des violations de clé d'index. Dans ce cas, ces éléments restent dans la table, mais l'index ne contient pas d'entrée correspondante.

Une violation de clé d'index se produit dans les situations suivantes :

- Il existe une discordance de type de données entre une valeur d'attribut et le type de données du schéma de clé d'index. Par exemple, supposons que l'un des éléments de la table `GameScores` a une valeur `TopScore` de type `String`. Si vous ajoutiez un index secondaire global avec une clé de partition `TopScore` de type `Number`, l'élément de la table violerait la clé d'index.
- Une valeur d'attribut de la table dépasse la longueur maximum pour un attribut de clé d'index. La longueur maximum d'une clé de partition est de 2 048 octets, et la longueur maximum d'une clé de tri est de 1 024 octets. Si l'une des valeurs d'attribut correspondantes dans la table dépasse ces limites, l'élément de la table viole la clé d'index.

> **Note**
>
> Si une valeur de String ou de Binary attribute est définie pour un attribut utilisé comme clé d'index, la valeur d'attribut doit avoir une longueur supérieure à zéro ; sinon, l'élément de la table viole la clé d'index.
> Cet outil ne signale pas cette violation de clé d'index, pour le moment.

Si une violation de clé d'index se produit, la phase de remplissage se poursuit sans interruption. Toutefois, les éléments violant le schéma de clé ne sont pas inclus dans l'index. Une fois la phase de remplissage terminée, toutes les écritures dans des éléments, qui violent le schéma de clé du nouvel index sont rejetées.

Pour identifier et corriger les valeurs d'attribut dans une table, qui violent une clé d'index, utilisez l'outil de détection des violations. Pour exécuter l'outil de détection des violations, vous créez un fichier de configuration qui spécifie le nom d'une table à analyser, les noms et les types de données des clés de partition et de tri d'index secondaire global, ainsi que les actions à entreprendre en cas détection de violations de clé d'index. L'outil de détection des violations peut opérer dans l'un des deux modes suivants :

- Mode détection – Détecter les violations de clé d'index. Utilisez le mode de détection pour signaler les éléments de la table qui provoqueraient des violations de clé dans un index secondaire global (vous pouvez éventuellement demander que ces éléments de table violant le schéma de clé soient supprimés immédiatement quand ils sont détectés). La sortie du mode détection est écrite dans un fichier que vous pouvez utiliser pour une analyse plus approfondie.
- Mode correction – Corriger les violations de clé d'index. En mode correction, l'outil de détection des violations lit un fichier d'entrée du même format que le fichier de sortie du mode détection. Le mode correction lit les enregistrements du fichier d'entrée et, pour chacun d'eux, supprime ou met à jour les éléments correspondants dans la table (notez que, si vous choisissez de mettre à jour les éléments, vous devez modifier le fichier d'entrée et définir des valeurs appropriées pour ces mises à jour).

Téléchargement et exécution de l'outil de détection des violations

L'outil de détection des violations est disponible en tant qu'archive Java exécutable (fichier `.jar`), et s'exécute sur les ordinateurs Windows, macOS ou Linux. L'outil de détection des violations requiert Java 1.7 (ou version ultérieure) et Apache Maven.

- Télécharger l'outil de détection des violations à partir de GitHub

Suivez les instructions figurant dans le fichier `README.md` pour télécharger et installer l'outil de détection des violations à l'aide de Maven.

Pour démarrer l'outil de détection des violations, accédez au répertoire dans lequel vous avez généré le fichier `ViolationDetector.java`, puis entrez la commande suivante.

```
java -jar ViolationDetector.jar [options]
```

La ligne de commande de l'outil de détection des violations accepte les options suivantes :

- `-h | --help` – Affiche un résumé et des options d'utilisation pour l'outil de détection des violations.
- `-p | --configFilePath value` – Nom complet d'un fichier de configuration de l'outil de détection des violations. Pour plus d'informations, consultez Fichier de configuration de l'outil de détection des violations.
- `-t | --detect value` – Détecte les violations de clé d'index dans la table, et les écris dans le fichier de sortie de l'outil de détection des violations. Si la valeur de ce paramètre est définie sur

`keep`, les éléments comportant des violations de clé ne sont pas modifiés. Si la valeur est définie sur `delete`, les éléments comportant des violations de clé sont supprimés de la table.

- `-c | --correct value` – Lit les violations de clé d'index à partir d'un fichier d'entrée, et prend des mesures correctives sur les éléments de la table. Si la valeur de ce paramètre est définie sur `update`, les éléments comportant des violations de clé sont mis à jour avec de nouvelles valeurs ne violant pas le schéma de clé. Si la valeur est définie sur `delete`, les éléments comportant des violations de clé sont supprimés de la table.

Fichier de configuration de l'outil de détection des violations

Lors de l'exécution, l'outil de détection des violations nécessite un fichier de configuration. Les paramètres de ce fichier déterminent les ressources DynamoDB auxquelles l'outil de détection des violations peut accéder, ainsi que le débit approvisionné qu'il peut consommer. Le tableau suivant décrit ces paramètres.

Nom du paramètre	Description	Obligatoire ?

`awsCredentialsFile`	Le nom complet d'un fichier contenant vos informations d'identification AWS. Le format du fichier contenant les informations d'identification doit être le suivant :
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Oui

`dynamoDBRegion`	La région AWS dans laquelle la table réside. Par exemple : `us-west-2` .	Oui
`tableName`	Le nom de la table DynamoDB à analyser.	Oui

Nom du paramètre	Description	Obligatoire ?
`gsiHashKeyName`	Le nom de la clé de partition d'index.	Oui
`gsiHashKeyType`	Le type de données de la clé de partition d'index, à savoir `String`, `Number` ou `Binary` : `S | N | B`	Oui
`gsiRangeKeyName`	Le nom de la clé de tri d'index. Ne spécifiez pas ce paramètre si l'index n'a qu'une clé primaire simple (clé de partition).	Non
`gsiRangeKeyType`	Le type de données de la clé de tri d'index, à savoir `String`, `Number` ou `Binary` : `S | N | B` Ne spécifiez pas ce paramètre si l'index n'a qu'une clé primaire simple (clé de partition).	Non

Nom du paramètre	Description	Obligatoire ?
`recordDetails`	Indication relative à l'écriture des détails des violations de clé d'index dans le fichier de sortie. Si la valeur est définie sur `true` (par défaut), les informations complètes sur les éléments violant le schéma de clé sont rapportées. Si la valeur est définie sur `false`, seul le nombre de violations est rapporté.	Non
`recordGsiValueInViolationRecord`	Indication relative à l'écriture des valeurs des violations de clé d'index dans le fichier de sortie. Si la valeur est définie sur `true` (par défaut), les valeurs de clés sont rapportées. Si la valeur est définie sur `false`, les valeurs de clés sont rapportées.	Non

Nom du paramètre	Description	Obligatoire ?

`detectionOutputPath`	Le chemin d'accès complet du fichier de sortie de l'outil de détection des violations. Ce paramètre prend en charge l'écriture dans un répertoire local ou sur Amazon Simple Storage Service (Amazon S3). Voici quelques exemples :
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	Les informations figurant dans le fichier de sortie s'affichent au format CSV (valeurs séparées par des virgules). Si vous ne définissez pas `detectionOutputPath`, le fichier de sortie est nommé `violation_detection.csv` et écrit dans votre répertoire de travail actuel.
		Non

Nom du paramètre	Description	Obligatoire ?
`numOfSegments`	Le nombre de segments d'analyse parallèle à utiliser quand l'outil de détection des violations analyse la table. La valeur par défaut est 1, ce qui signifie que la table est analysée de manière séquentielle. Si la valeur est égale ou supérieure à 2, l'outil de détection des violations divise la table en plusieurs segments logiques et un nombre égal d'unités d'exécution d'analyse. La valeur maximum de `numOfSegments` est 4 096. Pour les tables plus volumineuses, une analyse parallèle est généralement plus rapide qu'une analyse séquentielle. En outre, si la table est suffisamment grande pour couvrir plusieurs partitions, une analyse parallèle répartit uniformément son activité de lecture sur plusieurs partitions. Pour plus d'informations sur les analyses parallèles dans DynamoDB, consultez Analyse parallèle.	Non

Nom du paramètre	Description	Obligatoire ?
`numOfViolations`	La limite supérieure des violations de clé d'index à écrire dans le fichier de sortie. Si la valeur est définie sur `-1` (par défaut), la table entière est analysée. Si la valeur est définie sur un nombre entier positif, l'outil de détection des violations s'arrête après avoir rencontré ce nombre de violations.	Non
`numOfRecords`	Le nombre d'éléments de la table à analyser. Si la valeur est définie sur (par défaut), la table entière est analysée. Si la valeur est définie sur un nombre entier positif, l'outil de détection des violations s'arrête après avoir analysé ce nombre d'éléments de la table.	Non
`readWriteIOPSPercent`	Régule le pourcentage d'unités de capacité de lecture approvisionnée qui sont consommées pendant l'analyse de la table. La plage des valeurs valides s'étend de `1` à `100`. La valeur par défaut (`25`) signifie que l'outil de détection des violation s ne consommera pas plus de 25 % du débit de lecture approvisionné de la table.	Non

Nom du paramètre	Description	Obligatoire ?
`correctionInputPath`	Chemin d'accès complet du fichier d'entrée de correctio n de l'outil de détection des violations. Si vous exécutez l'outil de détection des violations en mode correctio n, le contenu de ce fichier est utilisé pour modifier ou supprimer les éléments de données dans la table, qui violent l'index secondaire global. Le format du fichier `correctionInputPath` est identique à celui du fichier `detectionOutputPat h` . Cela vous permet de traiter la sortie du mode détection comme entrée en mode correction.	Non

Nom du paramètre	Description	Obligatoire ?

`correctionOutputPath`	Chemin d'accès complet du fichier de sortie de correction de l'outil de détection des violations. Ce fichier n'est créé que s'il y a des erreurs de mise à jour.
	
	Ce paramètre prend en charge l'écriture dans un répertoire local ou sur Amazon S3. Voici quelques exemples :
	
	`correctionOutputPath = //local/path/filename.csv`
	
	`correctionOutputPath = s3://bucket/filename.csv`
	
	Les informations figurant dans le fichier de sortie s'affichent au format CSV. Si vous ne définissez pas `correctionOutputPath`, le fichier de sortie est nommé `violation_update_errors.csv` et écrit dans votre répertoire de travail actuel.
		Non

Détection

Pour détecter les violations de clé d'index, utilisez l'outil de détection des violations avec l'option de ligne de commande `--detect`. Pour voir le fonctionnement de cette option, considérez le tableau

`ProductCatalog`. Voici la liste des éléments de la table. Seule la clé primaire (`Id`) et l'attribut `Price` sont affichés.

Id (clé primaire)	Prix
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Toutes les valeurs pour `Price` sont de type `Number`. Cependant, DynamoDB étant sans schéma, il est possible d'ajouter un élément avec une valeur `Price` non numérique. Par exemple, supposons que vous ajoutez un autre élément à la table `ProductCatalog`.

Id (clé primaire)	Prix
999	"Hello"

La table totalise maintenant neuf éléments.

Vous ajoutez à présent un nouvel index secondaire global à la table : `PriceIndex`. La clé primaire pour cet index est une clé de partition, `Price`, de type `Number`. Une fois l'index généré, il contient huit éléments, tandis que la table `ProductCatalog` en contient neuf. La raison de cette différence est que la valeur `"Hello"` est de type `String`, alors que `PriceIndex` a une clé primaire de type `Number`. La valeur `String` violant la clé d'index secondaire globale, elle n'est pas présente dans l'index.

Pour utiliser l'outil de détection des violations dans ce scénario, vous devez d'abord créer un fichier de configuration tel que le suivant.

```
# Properties file for violation detection tool configuration.
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

Ensuite, vous exécutez l'outil de détection des violations comme dans l'exemple suivant.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

Si le paramètre de configuration `recordDetails` est défini sur `true`, l'outil de détection des violations écrit les détails de chaque violation dans le fichier de sortie, comme dans l'exemple suivant.

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)

999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

Le fichier de sortie est au format CSV. La première ligne du fichier est un en-tête. Elle est suivie d'un registre par élément qui viole la clé d'index. Les champs de ces registres de violation sont les suivants :

- Table hash key (Clé de hachage de table) – Valeur de clé de partition de l'élément dans la table.
- Table range key (Clé de plage de table) – Valeur de clé de tri de l'élément dans la table.
- GSI hash key value (Valeur de clé de hachage GSI) – Valeur de clé de partition de l'index secondaire global.
- GSI hash key violation type (Type de violation de clé de hachage GSI) – `Type Violation` ou `Size Violation`.
- GSI hash key violation description (Description de violation de clé de hachage GSI) – Cause de la violation.
- GSI hash key update value (FOR USER) (Valeur de mise à jour de clé de hachage GSI (POUR UTILISATEUR)) – En mode correction, nouvelle valeur fournie par l'utilisateur pour l'attribut.
- GSI range key value (Valeur de clé de plage GSI) – Valeur de clé de tri de l'index secondaire global.
- GSI range key violation type (Type de violation de clé de plage GSI) – `Type Violation` ou `Size Violation`.
- GSI range key violation description (Description de violation de clé de plage GSI) – Cause de la violation.
- GSI range key update value (FOR USER) (Valeur de mise à jour de clé de plage GSI (POUR UTILISATEUR)) – En mode correction, nouvelle valeur fournie par l'utilisateur pour l'attribut.
- Delete blank attribute when updating (Y/N) (Supprimer un attribut vide lors de la mise à jour (O/N)) – En mode correction, détermine s'il faut supprimer (Y) ou conserver (N) l'élément violant le schéma de clé dans la table, mais uniquement si l'un des champs suivants est vide :
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Si l'un de ces champs n'est pas vide, `Delete Blank Attribute When Updating(Y/N)` n'a aucun effet.

> **Note**
>
> Le format de sortie peut varier en fonction du fichier de configuration et des options de ligne de commande. Par exemple, si la table possède une clé primaire simple (sans clé de tri), aucun champ de clé de tri ne figure dans la sortie.
> Il se peut que les registres de violation dans le fichier ne soient pas dans un ordre trié.

Correction

Pour corriger les violations de clé d'indexation, utilisez l'outil de détection des violations avec l'option de ligne de commande `--correct`. En mode correction, l'outil de détection des violations lit le fichier d'entrée spécifié par le paramètre `correctionInputPath`. Le format de ce fichier étant le même que celui du fichier `detectionOutputPath`, vous pouvez utiliser la sortie de la détection comme entrée pour la correction.

L'outil de détection des violations offre deux options pour corriger les violations de clé d'index :

- Delete violations (Supprimer les violations) – Supprimer les éléments de table qui ont des valeurs d'attribut violant le schéma de clé.
- Update violations (Mettre à jour les violations) – Mettre à jour les éléments de table en remplaçant les attributs violant le schéma de clé par des valeurs ne les violant pas.

Dans les deux cas, vous pouvez utiliser le fichier de sortie du mode détection comme entrée pour le mode correction.

En poursuivant avec l'exemple `ProductCatalog`, supposons que vous voulez supprimer de la table l'élément violant le schéma de clé. Pour ce faire, vous utilisez la ligne de commande suivante.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

À ce stade, vous êtes invité à confirmer si vous souhaitez supprimer les éléments violant le schéma de clé.

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
```

```
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

A présent, `ProductCatalog` et `PriceIndex` ont le même nombre d'éléments.

Utilisation d'index secondaires globaux : Java

Vous pouvez utiliser l'API Document d'AWS SDK pour Java pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires globaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index.

Voici les étapes courantes pour les opérations de table.

1. Créez une instance de la classe `DynamoDB`.
2. Fournissez les paramètres obligatoires et facultatifs pour l'opération en créant les objets de requête correspondants.
3. Appelez la méthode appropriée fournie par le client, que vous avez créée à l'étape précédente.

Rubriques

- Créer une table avec un index secondaire global
- Décrire une table avec un index secondaire global
- Interroger un index secondaire global
- Exemple : index secondaires globaux utilisant l'API de document de AWS SDK pour Java

Créer une table avec un index secondaire global

Vous pouvez créer des index secondaires globaux au moment où vous créez une table. Pour ce faire, utilisez `CreateTable` et fournissez vos spécifications pour un ou plusieurs index secondaires globaux. L'exemple de code Java suivant crée une table destinée à accueillir des données météorologiques. La clé de partition est `Location`, et la clé de tri `Date`. Un index secondaire global nommé `PrecipIndex` permet un accès rapide aux données de précipitations pour différents endroits.

Voici les étapes à suivre pour créer une table avec un index secondaire global à l'aide de l'API Document DynamoDB.

1. Créez une instance de la classe `DynamoDB`.

2. Créez une instance de la classe `CreateTableRequest` pour fournir l'information de requête.

   Vous devez fournir le nom de la table, sa clé primaire et les valeurs de débit approvisionné. Pour l'index secondaire global, vous devez fournir le nom de l'index, ses paramètres de débit approvisionné, les définitions d'attribut pour la clé de tri d'index, le schéma de clé pour l'index et la projection d'attribut.
3. Appelez la méthode `createTable` en fournissant l'objet de demande comme paramètre.

L'exemple de code Java suivant illustre les tâches précédentes. Le code crée une table (`WeatherData`) avec un index secondaire global (`PrecipIndex`). La clé de partition d'index est `Date`, et la clé de tri `Precipitation`. Tous les attributs de table sont projetés sur l'index. Les utilisateurs peuvent interroger cet index afin d'obtenir des données météorologiques pour une date particulière, en triant éventuellement les données par quantité de précipitations.

`Precipitation` n'étant pas un attribut de clé pour la table, il n'est pas obligatoire. Toutefois, les éléments `WeatherData` sans `Precipitation` n'apparaissent pas dans `PrecipIndex`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));

// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
```

```
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table.

Décrire une table avec un index secondaire global

Pour obtenir des informations concernant les index secondaires globaux sur une table, utilisez l'opération `DescribeTable`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

Voici les étapes à suivre pour accéder aux informations d'index secondaire global sur une table.

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table` pour représenter l'index que vous souhaitez utiliser.
3. Appelez la méthode `describe` sur l'objet `Table`.

L'exemple de code Java suivant illustre les tâches précédentes.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

Interroger un index secondaire global

Vous pouvez utiliser l'opération `Query` sur un index secondaire global de la même manière que vous utilisez l'opération `Query` sur une table. Vous devez spécifier le nom de l'index, les critères de requête pour la clé de partition d'index et la clé de tri (le cas échéant), ainsi que les attributs

que vous souhaitez renvoyer. Dans cet exemple, l'index est `PrecipIndex`. Il a une clé de partition `Date` et une clé de tri `Precipitation`. L'interrogation de l'index renvoie toutes les données météorologiques pour une date particulière, où les précipitations sont supérieures à zéro.

Voici les étapes à suivre pour interroger un index secondaire global à l'aide de l'API Document d'AWS SDK pour Java.

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table` pour représenter l'index que vous souhaitez utiliser.
3. Créez une instance de la classe `Index` pour l'index que vous souhaitez interroger.
4. Appelez la méthode `query` sur l'objet `Index`.

Nom d'attribut `Date` est un mot réservé DynamoDB. Par conséquent, vous devez utiliser un nom d'attribut d'expression comme espace réservé dans la `KeyConditionExpression`.

L'exemple de code Java suivant illustre les tâches précédentes.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
}
```

Exemple : index secondaires globaux utilisant l'API de document de AWS SDK pour Java

L'exemple de code Java suivant montre comment utiliser les index secondaires globaux. L'exemple crée une table nommée `Issues`, utilisable dans un système simple de suivi de bogues pour le développement de logiciels. La clé de partition est `IssueId`, et la clé de tri `Title`. Il y a trois index secondaires globaux sur cette table :

- `CreateDateIndex` – La clé de partition est `CreateDate`, et la clé de tri `IssueId`. En plus des clés de table, les attributs `Description` et `Status` sont projetés dans l'index.
- `TitleIndex` – La clé de partition est `Title`, et la clé de tri `IssueId`. Aucun attribut autre que les clés de table n'est projeté dans l'index.
- `DueDateIndex` – La clé de partition est `DueDate` et il n'y a pas de clé de tri. Tous les attributs de table sont projetés sur l'index.

Une fois la table `Issues` créée, le programme charge la table avec des données représentant des rapports de bogues logiciels. Il interroge ensuite les données à l'aide d'index secondaires globaux. Enfin, le programme supprime la table `Issues`.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
```

```
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";

    public static void main(String[] args) throws Exception {

        createTable();
        loadData();

        queryIndex("CreateDateIndex");
        queryIndex("TitleIndex");
        queryIndex("DueDateIndex");

        deleteTable(tableName);

    }

    public static void createTable() {

        // Attribute definitions
        ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

        // Key schema for table
        ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
```

```
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
Partition

                    // key
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                   // key

        // Initial provisioned throughput settings for the indexes
        ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
                .withWriteCapacityUnits(1L);

        // CreateDateIndex
        GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                          // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(
                        new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));

        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                     // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort
```

```
                // key
               .withProjection(new Projection().withProjectionType("KEYS_ONLY"));

       // DueDateIndex
       GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
               .withProvisionedThroughput(ptIndex)
               .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                     // key
               .withProjection(new Projection().withProjectionType("ALL"));

       CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
               .withProvisionedThroughput(
                       new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
               .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

       System.out.println("Creating table " + tableName + "...");
       dynamoDB.createTable(createTableRequest);

       // Wait for table to become active
       System.out.println("Waiting for " + tableName + " to become ACTIVE...");
       try {
           Table table = dynamoDB.getTable(tableName);
           table.waitForActive();
       } catch (InterruptedException e) {
           e.printStackTrace();
       }
   }

   public static void queryIndex(String indexName) {

       Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
       System.out.print("Querying index " + indexName + "...");
```

```
        Index index = table.getIndex(indexName);

        ItemCollection<QueryOutcome> items = null;

        QuerySpec querySpec = new QuerySpec();

        if (indexName == "CreateDateIndex") {
            System.out.println("Issues filed on 2013-11-01");
            querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "TitleIndex") {
            System.out.println("Compilation errors");
            querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                    .withValueMap(
                            new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "DueDateIndex") {
            System.out.println("Items that are due on 2013-11-30");
            querySpec.withKeyConditionExpression("DueDate = :v_date")
                    .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
            items = index.query(querySpec);
        } else {
            System.out.println("\nNo valid index name provided");
            return;
        }

        Iterator<Item> iterator = items.iterator();

        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {
```

```
        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();

        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");

        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }
```

```
    public static void putItem(

            String issueId, String title, String description, String createDate, String
 lastUpdateDate, String dueDate,
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
 title)
                .withString("Description", description).withString("CreateDate",
 createDate)
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

Utilisation d'index secondaires globaux : .NET

Vous pouvez utiliser l'API de bas niveau d'AWS SDK pour .NET pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires globaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index. Ces opérations mappent aux opérations DynamoDB correspondantes. Pour plus d'informations, consultez la Référence d'API Amazon DynamoDB.

Voici les étapes courantes pour les opérations de table à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires et facultatifs pour l'opération en créant les objets de requête correspondants.

   Par exemple, créez un objet `CreateTableRequest` pour créer une table, et un objet `QueryRequest` pour interroger une table ou un index.
3. Exécutez la méthode appropriée fournie par le client, que vous avez créée à l'étape précédente.

Rubriques

- Créer une table avec un index secondaire global
- Décrire une table avec un index secondaire global
- Interroger un index secondaire global
- Exemple : index secondaires globaux utilisant l'API de bas niveau AWS SDK pour .NET

Créer une table avec un index secondaire global

Vous pouvez créer des index secondaires globaux au moment où vous créez une table. Pour ce faire, utilisez `CreateTable` et fournissez vos spécifications pour un ou plusieurs index secondaires globaux. L'exemple de code C# suivant crée une table destinée à accueillir des données météorologiques. La clé de partition est `Location`, et la clé de tri `Date`. Un index secondaire global nommé `PrecipIndex` permet un accès rapide aux données de précipitations pour différents endroits.

Voici les étapes à suivre pour créer une table avec un index secondaire global à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Créez une instance de la classe `CreateTableRequest` pour fournir l'information de requête.

   Vous devez fournir le nom de la table, sa clé primaire et les valeurs de débit approvisionné. Pour l'index secondaire global, vous devez fournir le nom de l'index, ses paramètres de débit approvisionné, les définitions d'attribut pour la clé de tri d'index, le schéma de clé pour l'index et la projection d'attribut.
3. Exécutez la méthode `CreateTable` en fournissant l'objet de demande comme paramètre.

L'exemple de code C# suivant présente les étapes précédentes. Le code crée une table (`WeatherData`) avec un index secondaire global (`PrecipIndex`). La clé de partition d'index est `Date`, et la clé de tri `Precipitation`. Tous les attributs de table sont projetés sur l'index. Les utilisateurs peuvent interroger cet index afin d'obtenir des données météorologiques pour une date particulière, en triant éventuellement les données par quantité de précipitations.

`Precipitation` n'étant pas un attribut de clé pour la table, il n'est pas obligatoire. Toutefois, les éléments `WeatherData` sans `Precipitation` n'apparaissent pas dans `PrecipIndex`.

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";
```

```
// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
```

```
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;

CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table.

Décrire une table avec un index secondaire global

Pour obtenir des informations concernant les index secondaires globaux sur une table, utilisez l'opération `DescribeTable`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

Voici les étapes à suivre pour accéder aux informations d'index secondaire global pour une table à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Exécutez la méthode `describeTable` en fournissant l'objet de demande comme paramètre.

   Créez une instance de la classe `DescribeTableRequest` pour fournir l'information de requête. Vous devez fournir le nom de la table.

3.

L'exemple de code C# suivant présente les étapes précédentes.

Example

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

Interroger un index secondaire global

Vous pouvez utiliser l'opération `Query` sur un index secondaire global de la même manière que vous utilisez l'opération `Query` sur une table. Vous devez spécifier le nom de l'index, les critères de requête pour la clé de partition d'index et la clé de tri (le cas échéant), ainsi que les attributs que vous souhaitez renvoyer. Dans cet exemple, l'index est `PrecipIndex`. Il a une clé de partition `Date` et une clé de tri `Precipitation`. L'interrogation de l'index renvoie toutes les données météorologiques pour une date particulière, où les précipitations sont supérieures à zéro.

Voici les étapes à suivre pour interroger un index secondaire global à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Créez une instance de la classe `QueryRequest` pour fournir l'information de requête.
3. Exécutez la méthode `query` en fournissant l'objet de demande comme paramètre.

Nom d'attribut `Date` est un mot réservé DynamoDB. Par conséquent, vous devez utiliser un nom d'attribut d'expression comme espace réservé dans la `KeyConditionExpression`.

L'exemple de code C# suivant présente les étapes précédentes.

Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};

var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
```

```
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
}
```

Exemple : index secondaires globaux utilisant l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant montre comment utiliser des index secondaires globaux. L'exemple crée une table nommée `Issues`, utilisable dans un système simple de suivi de bogues pour le développement de logiciels. La clé de partition est `IssueId`, et la clé de tri `Title`. Il y a trois index secondaires globaux sur cette table :

- `CreateDateIndex` – La clé de partition est `CreateDate`, et la clé de tri `IssueId`. En plus des clés de table, les attributs `Description` et `Status` sont projetés dans l'index.
- `TitleIndex` – La clé de partition est `Title`, et la clé de tri `IssueId`. Aucun attribut autre que les clés de table n'est projeté dans l'index.
- `DueDateIndex` – La clé de partition est `DueDate` et il n'y a pas de clé de tri. Tous les attributs de table sont projetés sur l'index.

Une fois la table `Issues` créée, le programme charge la table avec des données représentant des rapports de bogues logiciels. Il interroge ensuite les données à l'aide d'index secondaires globaux. Enfin, le programme supprime la table `Issues`.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
```

```
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        public static String tableName = "Issues";

        public static void Main(string[] args)
        {
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };
```

```
            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
            {
                new KeySchemaElement {
                    AttributeName = "Title",
                    KeyType = "RANGE" //Sort key
                }
            }
        };

            // Initial provisioned throughput settings for the indexes
            var ptIndex = new ProvisionedThroughput
            {
                ReadCapacityUnits = 1L,
                WriteCapacityUnits = 1L
            };

            // CreateDateIndex
            var createDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "CreateDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "CreateDate", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "INCLUDE",
                    NonKeyAttributes = {
                    "Description", "Status"
                }
                }
            };
```

```
            // TitleIndex
            var titleIndex = new GlobalSecondaryIndex()
            {
                IndexName = "TitleIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "Title", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "KEYS_ONLY"
                }
            };

            // DueDateIndex
            var dueDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "DueDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "DueDate",
                    KeyType = "HASH" //Partition key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "ALL"
                }
            };



            var createTableRequest = new CreateTableRequest
            {
                TableName = tableName,
                ProvisionedThroughput = new ProvisionedThroughput
                {
```

```
                ReadCapacityUnits = (long)1,
                WriteCapacityUnits = (long)1
            },
            AttributeDefinitions = attributeDefinitions,
            KeySchema = tableKeySchema,
            GlobalSecondaryIndexes = {
            createDateIndex, titleIndex, dueDateIndex
        }
        };

        Console.WriteLine("Creating table " + tableName + "...");
        client.CreateTable(createTableRequest);

        WaitUntilTableReady(tableName);
    }

    private static void LoadData()
    {
        Console.WriteLine("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error",
            "Can't compile Project X - bad version number. What does this mean?",
            "2013-11-01", "2013-11-02", "2013-11-10",
            1, "Assigned");

        putItem("A-102", "Can't read data file",
            "The main data file is missing, or the permissions are incorrect",
            "2013-11-01", "2013-11-04", "2013-11-30",
            2, "In progress");

        putItem("A-103", "Test failure",
            "Functional test of Project X produces errors",
            "2013-11-01", "2013-11-02", "2013-11-10",
            1, "In progress");

        putItem("A-104", "Compilation error",
            "Variable 'messageCount' was not initialized.",
            "2013-11-15", "2013-11-16", "2013-11-30",
            3, "Assigned");
```

```
            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
            String issueId, String title,
            String description,
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
 AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
```

```
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
                {
                    TableName = tableName,
                    Item = item
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n*******************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
            Dictionary<string, AttributeValue> expressionAttributeValues = new
 Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
 begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
```

```
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
            else if (indexName == "TitleIndex")
            {
                Console.WriteLine("Compilation errors\n");

                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo valid index name provided");
                return;
            }
```

```
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "Priority")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
            }
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
            client.DeleteTable(new DeleteTableRequest
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        private static void WaitUntilTableReady(string tableName)
        {
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
```

```
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Gestion des index secondaires globaux dans DynamoDB à l'aide de l'AWS CLI

Vous pouvez utiliser l'AWS CLI pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires globaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index.

Rubriques

- Créer une table avec un index secondaire global
- Ajouter un index secondaire global à une table existante
- Décrire une table avec un index secondaire global
- Interroger un index secondaire global

Créer une table avec un index secondaire global

Vous pouvez créer des index secondaires globaux au moment où vous créez une table. Pour ce faire, utilisez le paramètre `create-table` et fournissez vos spécifications pour un ou plusieurs index secondaires globaux. L'exemple suivant crée une table nommée `GameScores` avec un index secondaire global nommé `GameTitleIndex`. La table de base a une clé de partition `UserId` et une clé de tri `GameTitle`, vous permettant de trouver efficacement le meilleur score d'un utilisateur pour un jeu spécifique, tandis que l'index secondaire global (GSI) a une clé de partition `GameTitle` et une clé de tri `TopScore`, vous permettant de trouver rapidement le score le plus élevé pour un jeu particulier.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
```

```
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table. Vous pouvez utiliser la commande describe-table pour déterminer l'état de la création de table.

Ajouter un index secondaire global à une table existante

Vous pouvez également ajouter ou modifier des index secondaires globaux après la création de la table. Pour ce faire, utilisez le paramètre `update-table` et fournissez vos spécifications pour un ou plusieurs index secondaires globaux. L'exemple suivant utilise le même schéma que l'exemple précédent, mais part du principe que la table a déjà été créée et que nous ajoutons le GSI par la suite.

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

Décrire une table avec un index secondaire global

Pour obtenir des informations concernant les index secondaires globaux sur une table, utilisez le paramètre `describe-table`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

```
aws dynamodb describe-table --table-name GameScores
```

Interroger un index secondaire global

Vous pouvez utiliser l'opération `query` sur un index secondaire global de la même manière que vous utilisez l'opération `query` sur une table. Vous devez spécifier le nom d'index, les critères de requête pour la clé de tri d'index et les attributs que vous souhaitez renvoyer. Dans cet exemple, l'index est `GameTitleIndex` et la clé de tri d'index est `GameTitle`.

Les seuls attributs renvoyés sont ceux qui ont été projetés dans l'index. Vous pourriez également modifier cette requête pour sélectionner des attributs autres que de clé, mais cela nécessiterait une activité d'extraction de table relativement coûteuse. Pour plus d'informations sur les extractions de table, consultez Projections d'attribut.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Index locaux secondaires

Certaines applications doivent uniquement interroger les données à l'aide de la clé primaire de la table de base. Cependant, il peut y avoir des situations où une clé de tri alternative serait utile. Pour donner à votre application un choix de clés de tri, vous pouvez créer un ou plusieurs index secondaires locaux sur une table Amazon DynamoDB, et émettre des demandes `Query` ou `Scan` par rapport à ces index.

Rubriques

- Étape 6 : Utilisation d'un index secondaire local
- Projections d'attribut
- Création d'un index secondaire local

- Lecture de données à partir d'un index secondaire local
- Écritures d'élément et index secondaires locaux
- Considérations relatives au débit alloué pour les index secondaires locaux
- Considérations relatives au stockage pour les index secondaires locaux
- Collections d'articles dans les index secondaires locaux
- Utilisation d'index secondaires locaux : Java
- Utilisation d'index secondaires locaux : .NET
- Utilisation d'index secondaires locaux dans la AWS CLI de DynamoDB

Étape 6 : Utilisation d'un index secondaire local

À titre d'exemple, considérez la table `Thread`. Cette table est utile pour une application telle que les Forums de discussion AWS. Le schéma suivant illustre la façon dont les éléments de la table seront organisés. (Tous les attributs ne sont pas affichés.)

Thread

ForumName	Subject	LastPostDateTime	Replies	
"S3"	"aaa"	"2015-03-15:17:24:31"	12	...
"S3"	"bbb"	"2015-01-22:23:18:01"	3	...
"S3"	"ccc"	"2015-02-31:13:14:21"	4	...
"S3"	"ddd"	"2015-01-03:09:21:11"	9	...
"EC2"	"yyy"	"2015-02-12:11:07:56"	18	...
"EC2"	"zzz"	"2015-01-18:07:33:42"	0	...
"RDS"	"rrr"	"2015-01-19:01:13:24"	3	...
"RDS"	"sss"	"2015-03-11:06:53:00"	11	...
"RDS"	"ttt"	"2015-10-22:12:19:44"	5	...
...	...	...	...	

DynamoDB stocke tous les éléments ayant la même valeur de clé de partition de manière continue. Dans cet exemple, étant donné un `ForumName` particulier, une opération `Query` pourrait localiser immédiatement toutes les unités d'exécution pour ce forum. Dans un groupe d'éléments ayant la même valeur de clé de partition, les éléments sont triés par valeur de clé de tri. Si la clé de tri (`Subject`) est également fournie dans la requête, DynamoDB peut réduire les résultats renvoyés, par exemple, en renvoyant toutes les unités d'exécution du forum « S3 » qui ont un `Subject` commençant par la lettre « a ».

Certaines demandes peuvent nécessiter des modèles d'accès aux données plus complexes. Par exemple :

- Quelles unités d'exécution de forum obtiennent le plus de vues et de réponses ?
- Quelle unité d'exécution dans un forum particulier a le plus grand nombre de messages ?
- Combien d'unités d'exécution ont été publiées dans un forum particulier au cours d'une période donnée ?

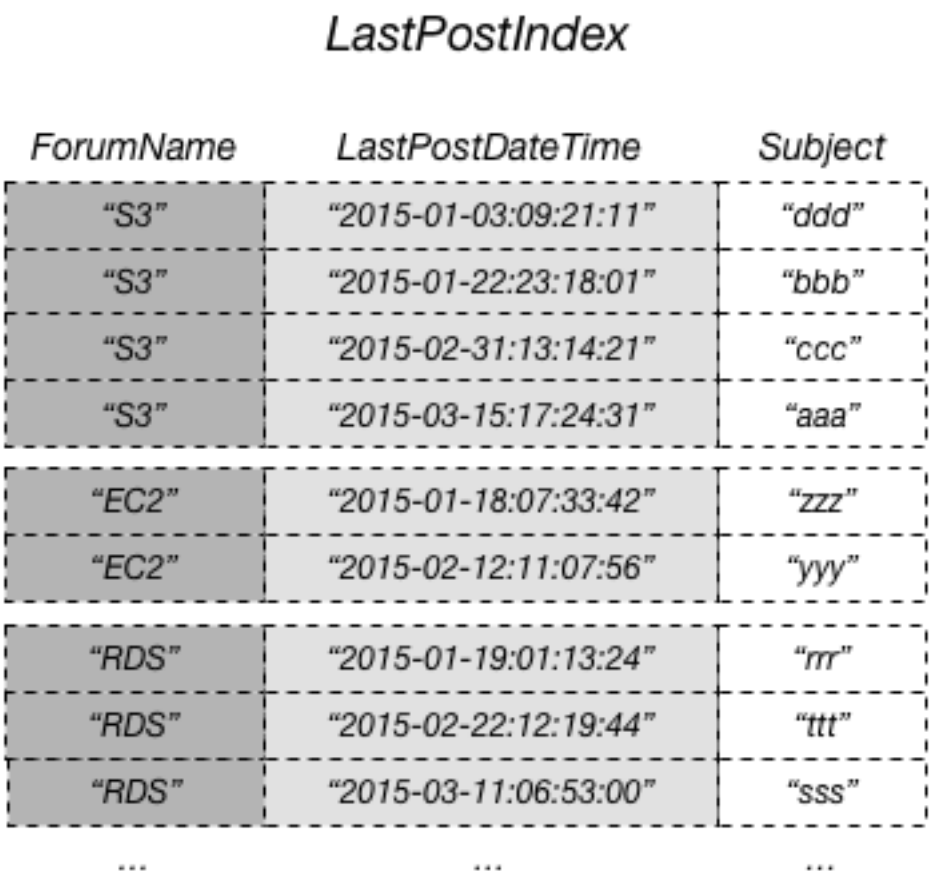
Pour répondre à ces questions, l'action `Query` ne serait pas suffisante. Au lieu de cela, vous devriez effectuer une opération `Scan` sur la table toute entière. Pour une table contenant des millions d'éléments, cette opération utiliserait une grande quantité de débit de lecture approvisionné et prendrait beaucoup de temps.

En revanche, vous pouvez spécifier un ou plusieurs index secondaires locaux sur des attributs autres que de clé, tels que `Replies` ou `LastPostDateTime`.

A index secondaire local gère une clé de tri alternative pour une valeur de clé de partition donnée. Un index secondaire local contient également une copie de tout ou partie des attributs de sa table de base. Vous spécifiez quels attributs sont projetés dans l'index secondaire local lorsque vous créez la table. Les données dans un index secondaire local sont organisées avec la même clé de partition que la table de base, mais une clé de tri différente. Cela vous permet d'accéder efficacement aux éléments de données dans cette dimension différente. Pour bénéficier d'une plus grande flexibilité de requête ou d'analyse, vous pouvez créer jusqu'à cinq index secondaires locaux par table.

Supposons qu'une application ait besoin de trouver toutes les discussions publiées au cours des trois derniers mois dans un forum particulier. A défaut d'index secondaire local, l'application devrait effectuer une opération `Scan` sur la table `Thread` toute entière, et écarter les publications ne d'inscrivant pas dans la période spécifiée. Avec un index secondaire local, une opération `Query` pourrait utiliser `LastPostDateTime` comme clé de tri et trouver les données rapidement.

Le diagramme suivant illustre un index secondaire local nommé `LastPostIndex`. Notez que la clé de partition est la même que celle de la table `Thread`, mais la clé de tri est `LastPostDateTime`.



Chaque index secondaire local doit remplir les conditions suivantes :

- La clé de partition est la même que celle de sa table de base.
- La clé de tri se compose exactement d'un attribut scalaire.
- La clé de tri de la table de base est projetée dans l'index, où elle agit comme un attribut autre que de clé.

Dans cet exemple, la clé de partition est `ForumName` et la clé de tri de l'index secondaire local est `LastPostDateTime`. En outre, la valeur de clé de tri de la table de base (dans cet exemple, `Subject`) est projetée dans l'index, mais ne fait pas partie de la clé d'index. Si une application a besoin d'une liste basée sur `ForumName` et `LastPostDateTime`, elle peut émettre une demande `Query` par rapport à `LastPostIndex`. Les résultats de la requête sont triés par `LastPostDateTime`, et peuvent être renvoyés dans un ordre croissant ou décroissant. La requête peut également appliquer des conditions de clé, telles que renvoyer uniquement les éléments qui ont une valeur `LastPostDateTime` s'inscrivant dans une période particulière.

Chaque index secondaire local contient automatiquement les clés de partition et de tri de sa table de base. Vous pouvez éventuellement projeter des attributs autres que de clé dans l'index. Lorsque vous interrogez l'index, DynamoDB peut extraire ces attributs projetés de façon efficace. Quand vous interrogez un index secondaire local, la requête peut également extraire des attributs qui ne sont pas projetés dans l'index. DynamoDB extrait automatiquement ces attributs de la table de base, mais moyennant une latence et des coûts de débit approvisionné plus élevés.

Pour tout index secondaire local, vous pouvez stocker jusqu'à 10 Go de données par valeur de clé de partition. Cette figure inclut tous les éléments de la table de base, ainsi que tous les éléments des index, qui ont la même valeur de clé de partition. Pour de plus amples informations, veuillez consulter Collections d'articles dans les index secondaires locaux.

Projections d'attribut

Avec `LastPostIndex`, une application pourrait utiliser `ForumName` et `LastPostDateTime` en tant que critères de requête. Toutefois, pour récupérer des attributs supplémentaires, DynamoDB doit effectuer des opérations de lecture supplémentaires par rapport à la table `Thread`. Ces lectures supplémentaires appelées extractions peuvent augmenter le volume total de débit approvisionné requis pour une requête.

Supposons que vous souhaitiez remplir une page Web avec une liste de tous les sujets de « S3 » et le nombre de réponses pour chaque fil, triés selon la dernière réponse en date/time commençant par la réponse la plus récente. Pour remplir cette liste, vous avez besoin des attributs suivants :

- `Subject`
- `Replies`
- `LastPostDateTime`

La manière la plus efficace d'interroger ces données et d'éviter les opérations d'extraction serait de projeter l'attribut `Replies` de la table vers l'index secondaire local, comme illustré dans ce diagramme.

LastPostIndex

ForumName	LastPostDateTime	Subject	Replies
"S3"	"2015-01-03:09:21:11"	"ddd"	9
"S3"	"2015-01-22:23:18:01"	"bbb"	3
"S3"	"2015-02-31:13:14:21"	"ccc"	4
"S3"	"2015-03-15:17:24:31"	"aaa"	12
"EC2"	"2015-01-18:07:33:42"	"zzz"	0
"EC2"	"2015-02-12:11:07:56"	"yyy"	18
"RDS"	"2015-01-19:01:13:24"	"rrr"	3
"RDS"	"2015-02-22:12:19:44"	"ttt"	5
"RDS"	"2015-03-11:06:53:00"	"sss"	11
...	...	...	...

Une projection est l'ensemble d'attributs copié à partir d'une table dans un index secondaire. Les clés de partition et de tri de la table sont toujours projetées dans l'index. Vous pouvez projeter d'autres attributs en fonction des exigences de requête de votre application. Lorsque vous interrogez un index, Amazon DynamoDB peut accéder à n'importe quel attribut dans la projection, comme s'il se trouvait dans une table.

Lorsque vous créez un index secondaire, vous devez spécifier les attributs qui seront projetés dans celui-ci. Pour cela, DynamoDB propose trois options :

- KEYS_ONLY – Chaque élément de l'index se compose uniquement des valeurs de clé de partition et de tri de la table, ainsi que des valeurs de clé d'index. L'option `KEYS_ONLY` produit l'index secondaire le plus petit possible.
- INCLUDE – En plus des attributs décrits dans `KEYS_ONLY`, l'index secondaire inclut des attributs autres que de clé, que vous spécifiez.
- ALL – L'index secondaire inclut tous les attributs de la table source. Toutes les données de la table étant dupliquées dans l'index, une projection `ALL` produit l'index secondaire le plus grand possible.

Dans le diagramme précédent, l'attribut autre que de clé `Replies` est projeté dans `LastPostIndex`. Une application peut interroger `LastPostIndex` au lieu de la table `Thread` toute entière pour remplir une page avec des sujets (`Subject`), réponses (`Replies`) et horodatages de dernière publication `LastPostDateTime`. Si d'autres attributs autres que de clé sont demandés, DynamoDB doit les extraire de la table `Thread`.

Du point de vue d'une application, l'extraction d'attributs supplémentaires à partir de la table de base est automatique et transparente. Il n'est donc pas nécessaire de réécrire une logique d'application. Cependant, une telle récupération peut réduire considérablement l'avantage en termes de performances de l'utilisation d'un index secondaire local.

Lorsque vous choisissez les attributs à projeter sur un index secondaire local, vous devez prendre en compte le compromis entre les coûts de débit approvisionné et les coûts de stockage :

- Si vous n'avez besoin d'accéder qu'à quelques attributs avec la plus faible latence possible, envisagez de projeter ces seuls attributs dans un index secondaire local. Plus l'index est petit, moins les coûts d'écriture et de stockage sont élevés. S'il existe des attributs que vous devez extraire occasionnellement, le coût du débit approvisionné risque d'être supérieur au coût à plus long terme du stockage de ces attributs.
- Si votre application accède fréquemment à certains attributs autres que de clé, vous devez envisager de projeter ces attributs dans un index secondaire local. Les coûts de stockage supplémentaires de l'index secondaire local compensent le coût d'exécution d'analyses de table fréquentes.
- Si vous avez besoin d'accéder fréquemment à la plupart des attributs autres que de clé, vous pouvez projeter ceux-ci (voire la table de base toute entière) dans un index secondaire local. Vous bénéficiez ainsi d'une flexibilité maximale et d'une consommation de débit approvisionné minimale, car aucune extraction n'est requise. Cependant, vos coûts de stockage augmentent, voire doublent, si vous projetez tous les attributs.
- Si votre application doit interroger une table peu fréquemment, mais doit effectuer un grand nombre d'écritures ou de mises à jour de données dans la table, pensez à projeter KEYS_ONLY. L'index secondaire local sera d'une taille minimum, mais continuera d'être disponible chaque fois que ce sera nécessaire pour une activité de requête.

Création d'un index secondaire local

Pour créer un ou plusieurs index secondaires locaux sur une table, utilisez le paramètre `LocalSecondaryIndexes` de l'opération `CreateTable`. Des index secondaires locaux sur une

table sont créés lors de la création de celle-ci. Lorsque vous supprimez une table, tous les index secondaires globaux sur celle-ci sont également supprimés.

Vous devez spécifier un attribut autre que de clé pour agir en tant que clé de tri de l'index secondaire local. L'attribut que vous choisissez doit être un scalaire `String`, `Number` ou `Binary`. Les autres types scalaires, types de document et types d'ensemble ne sont pas autorisés. Pour obtenir la liste complète des types de données, consultez Types de données.

> **Important**
>
> Pour les tables avec des index secondaires locaux, il existe une limite de taille de 10 Go par valeur de clé de partition. Une table avec des index secondaires locaux peut stocker n'importe quel nombre d'éléments, à condition qu'aucune valeur de clé de partition n'ait une taille supérieure à 10 Go. Pour de plus amples informations, veuillez consulter Taille limite de collection d'éléments.

Vous pouvez projeter des attributs de tout type de données dans un index secondaire local. Celui-ci inclut des scalaires, des documents et des ensembles. Pour obtenir la liste complète des types de données, consultez Types de données.

Lecture de données à partir d'un index secondaire local

Vous pouvez extraire des éléments d'un index secondaire local à l'aide des opérations `Query` et `Scan`. Vous ne pouvez pas utiliser les opérations `GetItem` et `BatchGetItem` sur un index secondaire local.

Interrogation d'un index secondaire local

Dans une table DynamoDB, les valeurs combinées des clés de partition et de tri pour chaque élément doivent être uniques. Cependant, dans un index secondaire local, la valeur de clé de tri n'a pas besoin d'être unique pour une valeur de clé de partition donnée. Si plusieurs éléments dans l'index secondaire local ont la même valeur de clé de tri, une opération `Query` renvoie tous les éléments ayant la même valeur de clé de partition. Dans la réponse, les éléments correspondants ne sont pas renvoyés dans un ordre particulier.

Vous pouvez interroger un index secondaire local en utilisant des lectures éventuellement ou fortement cohérentes. Pour spécifier le type de cohérence que vous souhaitez, utilisez le paramètre `ConsistentRead` de l'opération `Query`. Une lecture fortement cohérente d'un index secondaire

local renvoie toujours les dernières valeurs mises à jour. Si la requête doit extraire des attributs supplémentaires de la table de base, ces attributs sont cohérents par rapport à l'index.

Example

Considérez les données suivantes renvoyées par une opération `Query` qui demande des données des unités d'exécution de discussion dans un forum particulier.

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

Dans cette requête :

- DynamoDB y `LastPostIndex` accède en utilisant `ForumName` la clé de partition pour localiser les éléments d'index pour « ». EC2 Tous les éléments d'index avec cette clé sont stockés les uns à côté des autres pour une récupération rapide.
- Dans ce forum, DynamoDB utilise l'index pour rechercher les clés correspondant à la condition `LastPostDateTime` spécifiée.
- L'attribut `Replies` étant projeté dans l'index, DynamoDB peut extraire cet attribut sans utiliser de débit approvisionné supplémentaire.
- L'attribut `Tags` n'étant pas projeté dans l'index, DynamoDB doit accéder à la table `Thread` pour l'en extraire.
- Les résultats sont renvoyés, triés par `LastPostDateTime`. Les entrées d'index sont triées par valeur de clé de partition, puis par valeur de clé de tri, et `Query` les renvoie dans l'ordre de leur stockage (vous pouvez utiliser le paramètre `ScanIndexForward` pour renvoyer les résultats dans l'ordre décroissant).

L'attribut `Tags` n'étant pas projeté dans l'index secondaire local, DynamoDB doit utiliser des unités de capacité de lecture supplémentaires pour l'extraire de la table de base. Si vous devez exécuter cette requête souvent, nous vous recommandons de projeter `Tags` dans `LastPostIndex` pour éviter une extraction à partir de la table de base. Toutefois, si vous n'avez besoin d'accéder à `Tags` qu'occasionnellement, il se peut que le coût de stockage supplémentaire lié à la projection de `Tags` dans l'index n'en vaille pas la peine.

Analyse d'un index secondaire local

Vous pouvez utiliser l'opération `Scan` pour extraire toutes les données d'un index secondaire local. Vous devez fournir le nom de la table de base et le nom de l'index dans la demande. Avec une opération `Scan`, DynamoDB lit toutes les données de l'index et les renvoie à l'application. Vous pouvez également demander que seules quelques données soient retournées, et que les données restantes soient ignorées. Pour ce faire, utilisez le paramètre `FilterExpression` de l'API `Scan`. Pour de plus amples informations, veuillez consulter Filtrer les expressions à des fins d'analyse.

Écritures d'élément et index secondaires locaux

DynamoDB conserve automatiquement tous les index secondaires locaux synchronisés avec leurs tables de base respectives. Les applications n'écrivent jamais directement sur un index. Cependant, il est important de comprendre les implications de la façon dont DynamoDB gère ces index.

Lorsque vous créez un index secondaire local, vous spécifiez un attribut qui servira de clé de tri pour l'index. Vous spécifiez également un type de données pour cet attribut. Cela signifie que, chaque fois que vous écrivez un élément dans la table de base, si l'élément définit un attribut de clé d'index, son type doit correspondre au type de données du schéma de clé d'index. Dans le cas de `LastPostIndex`, la clé de tri `LastPostDateTime` dans l'index est définie comme un type de données `String`. Si vous essayez d'ajouter un élément à la table `Thread` et spécifiez un type de données différent pour `LastPostDateTime` (par exemple, `Number`), DynamoDB renvoie une `ValidationException` en raison de la discordance des types de données.

Il n'est pas nécessaire d' one-to-oneétablir une relation entre les éléments d'une table de base et ceux d'un index secondaire local. En fait, ce comportement peut être avantageux pour de nombreuses applications.

Une table avec de nombreux index secondaires locaux s'expose à des coûts plus élevés pour l'activité d'écriture qu'une table ayant moins d'index. Pour de plus amples informations, veuillez consulter Considérations relatives au débit alloué pour les index secondaires locaux.

> **Important**
>
> Pour les tables avec des index secondaires locaux, il existe une limite de taille de 10 Go par valeur de clé de partition. Une table avec des index secondaires locaux peut stocker n'importe quel nombre d'éléments, à condition qu'aucune valeur de clé de partition n'ait une taille supérieure à 10 Go. Pour de plus amples informations, veuillez consulter Taille limite de collection d'éléments.

Considérations relatives au débit alloué pour les index secondaires locaux

Lorsque vous créez une table dans DynamoDB, vous approvisionnez des unités de capacité de lecture et d'écriture pour la charge de travail prévue de la table. Cette charge de travail inclut l'activité de lecture et d'écriture sur les index secondaires locaux de la table.

Pour voir les prix actuels de capacité de débit approvisionnée, consultez Tarification Amazon DynamoDB.

Unités de capacité de lecture

Lorsque vous interrogez un index secondaire local, le nombre d'unités de capacité de lecture consommées dépend du mode d'accès aux données.

Comme une interrogation de table, une interrogation d'index peut utiliser des lectures éventuellement ou fortement cohérentes en fonction de la valeur de `ConsistentRead`. Une lecture fortement cohérente utilise une unité de capacité de lecture ; une lecture éventuellement cohérente n'en utilise que la moitié. Ainsi, en choisissant des lectures éventuellement cohérentes, vous pouvez réduire vos coûts d'unités de capacité de lecture.

Pour les interrogations d'index qui demandent uniquement des clés d'index et des attributs projetés, DynamoDB calcule l'activité de lecture approvisionnée de la même façon que pour des requêtes sur des tables. La seule différence est que le calcul est basé sur les tailles des entrées d'index, plutôt que sur la taille de l'élément de la table de base. Le nombre d'unités de capacité de lecture est la somme des tailles de tous les attributs projetés de tous les éléments renvoyés. Le résultat est ensuite arrondi à la limite de 4 Ko suivante. Pour plus d'informations sur la façon dont DynamoDB calcule l'utilisation du débit approvisionné, consultez Mode de capacité provisionnée DynamoDB.

Pour les interrogations d'index qui lisent des attributs qui ne sont pas projetés dans l'index secondaire local, DynamoDB doit extraire ces attributs de la table de base, en plus de lire les attributs projetés à

partir de l'index. Ces extractions se produisent lorsque vous incluez des attributs non projetés dans les paramètres `Select` ou `ProjectionExpression` de l'opération `Query`. L'extraction entraîne une latence supplémentaire des réponses aux requêtes, ainsi qu'un surcoût de débit approvisionné. En plus des lectures de l'index secondaire local décrites précédemment, vous êtes facturé pour les unités de capacité de lecture utilisées pour extraire chaque élément de la table de base. Ces frais ont trait à la lecture de chaque élément entier de la table, pas seulement des attributs demandés.

La taille maximum des résultats renvoyés par une opération `Query` est de 1 Mo. Cela inclut les tailles de tous les noms et valeurs d'attribut à travers l'ensemble des éléments retournés. Toutefois, si une requête sur un index secondaire local amène DynamoDB à extraire des attributs d'élément de la table de base, la taille maximum des données dans les résultats peut être inférieure. Dans ce cas, la taille du résultat est la somme de :

- La taille des éléments correspondants dans l'index, arrondie à la limite de 4 Ko suivante.
- La taille de chaque élément correspondant dans la table de base, chaque élément étant arrondi individuellement à la limite de 4 Ko suivante.

Avec cette formule, la taille maximum des résultats renvoyés par une opération Query est toujours de 1 Mo.

Par exemple, considérons une table où la taille de chaque élément est de 300 octets. Il existe un index secondaire local sur cette table, mais seuls 200 octets de chaque élément sont projetés dans l'index. Supposons à présent que vous effectuez une opération `Query` sur cet index, que la requête nécessite une extraction de table pour chaque élément, et que la requête renvoie 4 éléments. DynamoDB additionne les valeurs suivantes :

- La taille des éléments correspondants dans l'index : 200 octets × 4 éléments = 800 octets, valeur ensuite arrondie à 4 Ko.
- La taille de chaque élément correspondant dans la table de base : (300 octets, arrondis à 4 Ko) × 4 éléments = 16 Ko.

La taille totale des données dans le résultat est donc de 20 Ko.

Unités de capacité d'écriture

Quand un élément d'une table est ajouté, mis à jour ou supprimé, la mise à jour des index secondaires locaux utilise des unités de capacité d'écriture approvisionnée pour la table. Le coût total du débit alloué pour une écriture est la somme des unités de capacité d'écriture consommées par

l'écriture dans la table de base, et de celles consommées par la mise à jour des index secondaires locaux.

Le coût d'écriture d'un élément dans un index secondaire local dépend de plusieurs facteurs :

- Si vous écrivez un nouvel élément sur la table qui définit un attribut indexé, ou que vous mettez à jour un élément existant pour définir un attribut indexé précédemment non défini, une opération d'écriture est nécessaire pour insérer l'élément dans l'index.
- Si une mise à jour de la table change la valeur d'un attribut de clé indexé (de A en B), deux écritures sont requises, une pour supprimer l'élément précédent de l'index et une autre pour insérer le nouvel élément dans l'index.
- Si un élément était présent dans l'index, mais qu'une écriture sur la table a entraîné la suppression de l'attribut indexé, une écriture est nécessaire pour supprimer de l'index la projection de l'ancien élément.
- Si un élément n'est pas présent dans l'index avant ou après la mise à jour de l'élément, il n'y a aucun coût d'écriture supplémentaire pour l'index.

Tous ces facteurs partent du principe que la taille de chaque élément dans l'index est inférieure ou égale à la taille d'élément de 1 Ko pour le calcul des unités de capacité d'écriture. Les plus grandes entrées d'index nécessitent des unités de capacité en écriture supplémentaires. Vous pouvez réduire vos coûts d'écriture en considérant les attributs que vos requêtes ont besoin de renvoyer et en projetant uniquement ces attributs dans l'index.

Considérations relatives au stockage pour les index secondaires locaux

Quand une application écrit un élément dans une table, DynamoDB copie automatiquement le sous-ensemble approprié d'attributs vers les index secondaires locaux où ces attributs doivent apparaître. Votre AWS compte est débité pour le stockage de l'article dans la table de base ainsi que pour le stockage des attributs dans les index secondaires locaux de cette table.

La quantité d'espace utilisée par un élément de l'index est la somme des éléments suivants :

- La taille en octets de la clé primaire de la table de base (clé de partition et clé de tri)
- La taille en octets de l'attribut de clé d'index
- La taille en octets des attributs projetés (le cas échéant)
- 100 octets de surcharge par élément d'index

Pour estimer les besoins en stockage d'un index secondaire local, vous pouvez estimer la taille moyenne d'un élément de l'index, puis multiplier cette valeur par le nombre d'éléments présents dans l'index.

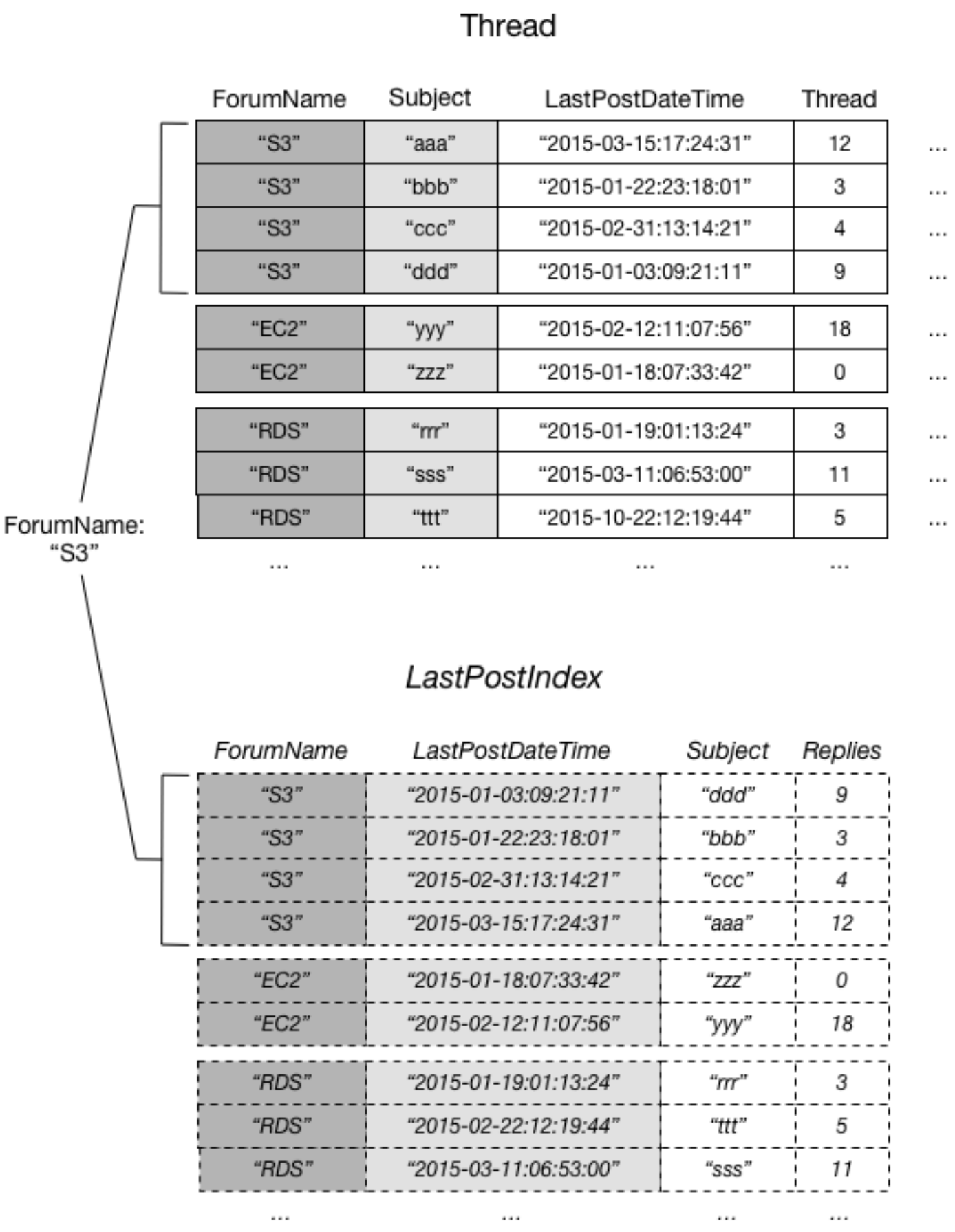
Si une table contient un élément dans lequel un attribut particulier n'est pas défini, tandis que cet attribut est défini en tant que clé de tri d'index, DynamoDB n'écrit aucune donnée pour cet élément dans l'index.

Collections d'articles dans les index secondaires locaux

> **Note**
>
> Cette section concerne uniquement les tables qui ont des index secondaires locaux.

Dans DynamoDB, une collection d'éléments est un groupe quelconque d'éléments ayant la même valeur de clé de partition dans une table, ainsi que de tous leurs index secondaires locaux. Dans les exemples utilisés tout au long de cette section, la clé de partition pour la table `Thread` est `ForumName`, et la clé de partition pour `LastPostIndex` est également `ForumName`. Tous les éléments de table et d'index ayant le même `ForumName` font partie de la même collection d'éléments. Par exemple, la table `Thread` et l'index secondaire local `LastPostIndex` contiennent une collection d'éléments pour le forum `EC2`, et une autre pour le forum `RDS`.

Le diagramme suivant illustre la collection d'éléments pour le forum `S3`.


Thread
ForumName
Subject
LastPostDateTime
Thread
"S3"
"aaa"
"2015-03-15:17:24:31"
12
"S3"
"bbb"
"2015-01-22:23:18:01"
3
"S3"
"ccc"
"2015-02-31:13:14:21"
4
"S3"
"ddd"
"2015-01-03:09:21:11"
9
"EC2"
"yyy"
"2015-02-12:11:07:56"
18
"EC2"
"zzz"
"2015-01-18:07:33:42"
0
"RDS"
"rrr"
"2015-01-19:01:13:24"
3
"RDS"
"sss"
"2015-03-11:06:53:00"
11
"RDS"
"ttt"
"2015-10-22:12:19:44"
5
ForumName:
"S3"
LastPostIndex
ForumName
LastPostDateTime
Subject
Replies
"S3"
"2015-01-03:09:21:11"
"ddd"
9
"S3"
"2015-01-22:23:18:01"
"bbb"
3
"S3"
"2015-02-31:13:14:21"
"ccc"
4
"S3"
"2015-03-15:17:24:31"
"aaa"
12
"EC2"
"2015-01-18:07:33:42"
"zzz"
0
"EC2"
"2015-02-12:11:07:56"
"yyy"
18
"RDS"
"2015-01-19:01:13:24"
"rrr"
3
"RDS"
"2015-02-22:12:19:44"
"ttt"
5
"RDS"
"2015-03-11:06:53:00"
"sss"
11

Dans ce diagramme, la collection d'éléments se compose de tous les éléments figurant dans `Thread` et `LastPostIndex`, où la valeur de clé de partition `ForumName` est « S3 ». S'il y avait d'autres index secondaires locaux sur la table, tous les éléments dans ces index dont la valeur de `ForumName` est « S3 » feraient également partie de la collection d'éléments.

Vous pouvez utiliser n'importe laquelle des opérations suivantes dans DynamoDB pour renvoyer des informations sur les collections d'éléments :

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Chacune de ces opérations prend en charge le paramètre `ReturnItemCollectionMetrics`. Lorsque vous définissez ce paramètre sur `SIZE`, vous pouvez afficher des informations sur la taille de chaque collection d'éléments dans l'index.

Example

Voici un exemple de sortie d'une opération `UpdateItem` sur la table `Thread`, avec la valeur `ReturnItemCollectionMetrics` définie sur `SIZE`. L'élément mis à jour avait une `ForumName` valeur de EC2 « », de sorte que la sortie inclut des informations sur cette collection d'éléments.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

L'objet `SizeEstimateRangeGB` montre que la taille de cette collection d'éléments est comprise entre 0 et 1 Go. DynamoDB mettant régulièrement à jour cette estimation de taille, les nombres pourraient être différents lors de la prochaine modification de l'élément.

Taille limite de collection d'éléments

La taille maximale de toute collection d'éléments pour une table possédant un ou plusieurs index secondaires locaux est de 10 Go. Cela ne s'applique pas aux collections d'éléments des tables sans index secondaires locaux, ni aux collections d'éléments dans des index secondaires globaux. Seules les tables ayant un ou plusieurs index secondaires locaux sont concernées.

Si une collection d'éléments dépasse la limite de 10 Go, DynamoDB renvoie une erreur `ItemCollectionSizeLimitExceededException`, et vous ne pouvez ni ajouter des éléments à la collection, ni augmenter la taille des éléments qu'elle contient (les opérations de lecture et d'écriture qui réduisent la taille de la collection d'éléments restent autorisées). Vous pouvez toujours ajouter des éléments à d'autres collections d'éléments.

Pour réduire la taille d'une collection d'éléments, vous pouvez effectuer l'une des opérations suivantes :

- Supprimer tous les éléments inutiles avec la valeur de clé de partition en question. Lorsque vous supprimez ces éléments de la table de base, DynamoDB supprime également toutes les entrées d'index ayant la même valeur de clé de partition.
- Mettre à jour les éléments en supprimant des attributs ou en réduisant la taille des attributs. Si ces attributs sont projetés dans des index secondaires locaux, DynamoDB réduit également la taille des entrées d'index correspondantes.
- Créer une table avec les mêmes clés de partition et de tri, puis déplacer les éléments de l'ancienne table vers la nouvelle. Cela peut être une bonne approche si une table contient des données historiques rarement utilisées. Vous pouvez également envisager d'archiver ces données historiques sur Amazon Simple Storage Service (Amazon S3).

Lorsque la taille totale de la collection d'éléments passe sous le seuil de 10 Go, vous pouvez à nouveau ajouter des éléments avec la même valeur de clé de partition.

En guise de bonne pratique, nous vous recommandons d'instrumenter votre application pour contrôler les tailles de vos collections d'éléments. Une façon de le faire consiste à définir le paramètre `ReturnItemCollectionMetrics` sur `SIZE` chaque fois que vous utilisez `BatchWriteItem`, `DeleteItem`, `PutItem` ou `UpdateItem`. Votre application doit examiner l'objet `ReturnItemCollectionMetrics` dans la sortie et journaliser un message d'erreur chaque fois qu'une collection d'éléments dépasse une limite de taille définie par l'utilisateur (par exemple, 8 Go). Définir une limite inférieure à 10 Go fournirait un système d'avertissement précoce qui vous

permettrait de savoir qu'une collection d'éléments approche de la limite dans le temps pour faire quelque chose à ce sujet.

Collections et partitions d'éléments

Dans une table avec un ou plusieurs index secondaires locaux, chaque collection d'éléments est stockée dans une partition. La taille totale d'une collection d'éléments de ce type est limitée à la capacité de cette partition, soit 10 Go. Pour une application dans laquelle le modèle de données inclut des collections d'éléments de taille illimitée, ou dans laquelle vous pouvez raisonnablement vous attendre à ce que certaines collections d'éléments dépassent 10 Go à l'avenir, envisagez plutôt d'utiliser un index secondaire global.

Vous devez concevoir vos applications de sorte que les données de table soient réparties uniformément entre des valeurs de clé de partition distinctes. Pour les tables avec des index secondaires locaux, vos applications ne doivent pas créer de « points chauds » d'activité de lecture et d'écriture au sein d'une seule collection d'éléments sur une seule partition.

Utilisation d'index secondaires locaux : Java

Vous pouvez utiliser l'API Document d'AWS SDK pour Java pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires locaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index.

Voici les étapes courantes pour les opérations de table à l'aide de l'API Document AWS SDK pour Java.

1. Créez une instance de la classe `DynamoDB`.
2. Fournissez les paramètres obligatoires et facultatifs pour l'opération en créant les objets de requête correspondants.
3. Appelez la méthode appropriée fournie par le client, que vous avez créée à l'étape précédente.

Rubriques

- Créer une table avec un index secondaire local
- Décrire une table avec un index secondaire local
- Interroger un index secondaire local
- Exemple : index secondaires locaux utilisant l'API de document de Java

Créer une table avec un index secondaire local

Vous devez créer des index secondaires locaux au moment où vous créez une table. Pour ce faire, utilisez la méthode `createTable` et fournissez vos spécifications pour un ou plusieurs index secondaires locaux. L'exemple de code Java suivant crée une table destinée à accueillir des informations sur des chansons dans une collection musicale. La clé de partition est `Artist`, et la clé de tri `SongTitle`. Un index secondaire, `AlbumTitleIndex`, facilite les requêtes par titre d'album.

Voici les étapes à suivre pour créer une table avec un index secondaire local à l'aide de l'API Document DynamoDB.

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `CreateTableRequest` pour fournir l'information de requête.

   Vous devez fournir le nom de la table, sa clé primaire et les valeurs de débit approvisionné. Pour l'index secondaire local, vous devez fournir le nom d'index, le nom et le type de données pour la clé de tri d'index, le schéma de clé pour l'index et la projection d'attribut.
3. Appelez la méthode `createTable` en fournissant l'objet de demande comme paramètre.

L'exemple de code Java suivant illustre les tâches précédentes. Le code crée une table (`Music`) avec un index secondaire sur l'attribut `AlbumTitle`. Les clés de partition et de tri de table, ainsi que la clé de tri d'index, sont les seuls attributs projetés dans l'index.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));

//AttributeDefinitions
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
```

```
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table.

Décrire une table avec un index secondaire local

Pour obtenir des informations concernant les index secondaires locaux sur une table, utilisez la méthode `describeTable`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

Voici les étapes à suivre pour accéder aux informations d'index secondaire local sur une table à l'aide de l'API Document AWS SDK pour Java.

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table`. Vous devez fournir le nom de la table.
3. Appelez la méthode `describeTable` sur l'objet `Table`.

L'exemple de code Java suivant illustre les tâches précédentes.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {

    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
```

```
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

Interroger un index secondaire local

Vous pouvez utiliser l'opération `Query` sur un index secondaire local de la même manière que vous utilisez l'opération `Query` sur une table. Vous devez spécifier le nom d'index, les critères de requête pour la clé de tri d'index et les attributs que vous souhaitez renvoyer. Dans cet exemple, l'index est `AlbumTitleIndex` et la clé de tri d'index est `AlbumTitle`.

Les seuls attributs renvoyés sont ceux qui ont été projetés dans l'index. Vous pourriez également modifier cette requête pour sélectionner des attributs autres que de clé, mais cela nécessiterait une activité d'extraction de table relativement coûteuse. Pour plus d'informations sur les extractions de table, consultez Projections d’attribut.

Voici les étapes à suivre pour interroger un index secondaire local à l'aide de l'API Document d'AWS SDK pour Java.

1. Créez une instance de la classe `DynamoDB`.
2. Créez une instance de la classe `Table`. Vous devez fournir le nom de la table.
3. Créez une instance de la classe `Index`. Vous devez fournir le nom d'index.
4. Appelez la méthode `query` de la classe `Index`.

L'exemple de code Java suivant illustre les tâches précédentes.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";
```

```
Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

Exemple : index secondaires locaux utilisant l'API de document de Java

L'exemple de code Java suivant montre comment utiliser les index secondaires locaux dans Amazon DynamoDB. L'exemple crée une table nommée `CustomerOrders` avec une clé de partition `CustomerId` et une clé de tri `OrderId`. Il y a deux index secondaires locaux sur cette table :

- `OrderCreationDateIndex` – La clé de tri est.`OrderCreationDate`, et les attributs suivants sont projetés dans l'index :
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex` – La clé de tri est `IsOpen` et tous les attributs de table sont projetés dans l'index.

Une fois la table `CustomerOrders` créée, le programme charge la table avec des données représentant des commandes de clients. Il interroge ensuite les données à l'aide d'index secondaires locaux. Enfin, le programme supprime la table `CustomerOrders`.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.PutItemOutcome;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;

public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";

        public static void main(String[] args) throws Exception {

                createTable();
                loadData();

                query(null);
                query("IsOpenIndex");
```

```
                query("OrderCreationDateIndex");

                deleteTable(tableName);

        }

        public static void createTable() {

                CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                                .withProvisionedThroughput(
                                                new
ProvisionedThroughput().withReadCapacityUnits((long) 1)

.withWriteCapacityUnits((long) 1));

                // Attribute definitions for table partition and sort keys
                ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

                // Attribute definition for index primary key attributes
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("OrderCreationDate")
                                                .withAttributeType("N"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

                createTableRequest.setAttributeDefinitions(attributeDefinitions);

                // Key schema for table
                ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
```

```
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                                // key

                createTableRequest.setKeySchema(tableKeySchema);

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();

                // OrderCreationDateIndex
                LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                                .withIndexName("OrderCreationDateIndex");

                // Key schema for OrderCreationDateIndex
                ArrayList<KeySchemaElement> indexKeySchema = new
ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                               // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("OrderCreationDate")
                                .withKeyType(KeyType.RANGE)); // Sort
                                                              // key

                orderCreationDateIndex.setKeySchema(indexKeySchema);

                // Projection (with list of projected attributes) for
                // OrderCreationDateIndex
                Projection projection = new
Projection().withProjectionType(ProjectionType.INCLUDE);
                ArrayList<String> nonKeyAttributes = new ArrayList<String>();
                nonKeyAttributes.add("ProductCategory");
                nonKeyAttributes.add("ProductName");
                projection.setNonKeyAttributes(nonKeyAttributes);

                orderCreationDateIndex.setProjection(projection);

                localSecondaryIndexes.add(orderCreationDateIndex);

                // IsOpenIndex
```

```
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                            // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(dynamoDB.createTable(createTableRequest));

                // Wait for table to become active
                System.out.println("Waiting for " + tableName + " to become
ACTIVE...");
                try {
                        Table table = dynamoDB.getTable(tableName);
                        table.waitForActive();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void query(String indexName) {

                Table table = dynamoDB.getTable(tableName);
```

```
System.out.println("\n***********************************************************\n");
                System.out.println("Querying table " + tableName + "...");

                QuerySpec querySpec = new
QuerySpec().withConsistentRead(true).withScanIndexForward(true)

.withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        System.out.println("\nUsing index: '" + indexName + "': Bob's
orders that are open.");
                        System.out.println("Only a user-specified list of attributes
are returned\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
and IsOpen = :v_isopen")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        System.out.println("\nUsing index: '" + indexName
                                        + "': Bob's orders that were placed after
01/31/2015.");
                        System.out.println("Only the projected attributes are returned
\n");
                        Index index = table.getIndex(indexName);
```

```
                        querySpec.withKeyConditionExpression(
                                        "CustomerId = :v_custid and OrderCreationDate
>= :v_orddate")
                                        .withValueMap(
                                                        new
ValueMap().withString(":v_custid", "bob@example.com")

.withNumber(":v_orddate",

20150131));

                        querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else {
                        System.out.println("\nNo index: All of Bob's orders, by
OrderId:\n");

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com"));

                        ItemCollection<QueryOutcome> items = table.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                }

        }

        public static void deleteTable(String tableName) {
```

```
                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + tableName + "...");
                table.delete();

                // Wait for table to be deleted
                System.out.println("Waiting for " + tableName + " to be deleted...");
                try {
                        table.waitForDelete();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void loadData() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 2)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150221)
                                .withString("ProductCategory", "Bike")
                                .withString("ProductName", "Super Mountain")
                                .withString("OrderStatus", "ORDER RECEIVED");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 3)
```

```
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
                                .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                .withString("ProductName", "Calm Before The Storm")
                                .withString("OrderStatus", "SHIPPING DELAY")
                                .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
```

```
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                                .withString("ProductCategory", "Hardware")
                                .withString("ProductName", "Extra Heavy Hammer")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 5)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
20150309).withString("ProductCategory", "Book")
                                .withString("ProductName", "How To
Cook").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "440185");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 6)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150318).withString("ProductCategory", "Luggage")
                                .withString("ProductName", "Really Big
Suitcase").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "893927");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 7)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
20150324).withString("ProductCategory", "Golf")
                                .withString("ProductName", "PGA Pro
II").withString("OrderStatus", "OUT FOR DELIVERY")
                                .withString("ShipmentTrackingId", "383283");

                putItemOutcome = table.putItem(item);
                assert putItemOutcome != null;
        }
```

```
}
```

Utilisation d'index secondaires locaux : .NET

Rubriques

- Créer une table avec un index secondaire local
- Décrire une table avec un index secondaire local
- Interroger un index secondaire local
- Exemple : index secondaires locaux utilisant l'API de bas niveau AWS SDK pour .NET

Vous pouvez utiliser l'API de bas niveau d'AWS SDK pour .NET pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires locaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index. Ces opérations sont mappées aux actions de l'API de bas niveau DynamoDB correspondantes. Pour de plus amples informations, consultez Exemples de code .NET.

Voici les étapes courantes pour les opérations de table à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Fournissez les paramètres obligatoires et facultatifs pour l'opération en créant les objets de requête correspondants.

   Par exemple, créez un objet `CreateTableRequest` pour créer une table, et un objet `QueryRequest` pour interroger une table ou un index.
3. Exécutez la méthode appropriée fournie par le client, que vous avez créée à l'étape précédente.

Créer une table avec un index secondaire local

Vous devez créer des index secondaires locaux au moment où vous créez une table. Pour ce faire, utilisez `CreateTable` et fournissez vos spécifications pour un ou plusieurs index secondaires locaux. L'exemple de code C# suivant crée une table destinée à accueillir des informations sur des chansons dans une collection musicale. La clé de partition est `Artist`, et la clé de tri `SongTitle`. Un index secondaire, `AlbumTitleIndex`, facilite les requêtes par titre d'album.

Voici les étapes à suivre pour créer une table avec un index secondaire local à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Créez une instance de la classe `CreateTableRequest` pour fournir l'information de requête.

   Vous devez fournir le nom de la table, sa clé primaire et les valeurs de débit approvisionné. Pour l'index secondaire local, vous devez fournir le nom d'index, le nom et le type de données de la clé de tri d'index, le schéma de clé pour l'index et la projection d'attribut.
3. Exécutez la méthode `CreateTable` en fournissant l'objet de demande comme paramètre.

L'exemple de code C# suivant présente les étapes précédentes. Le code crée une table (`Music`) avec un index secondaire sur l'attribut `AlbumTitle`. Les clés de partition et de tri de table, ainsi que la clé de tri d'index, sont les seuls attributs projetés dans l'index.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
{
    AttributeName = "Artist",
    AttributeType = "S"
});

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });
```

```
attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
    IndexName = "AlbumTitleIndex",
    KeySchema = indexKeySchema,
    Projection = projection
};

List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

CreateTableResponse result = client.CreateTable(createTableRequest);
```

```
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table.

Décrire une table avec un index secondaire local

Pour obtenir des informations concernant les index secondaires locaux sur une table, utilisez l'API `DescribeTable`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

Voici les étapes à suivre pour accéder aux informations d'index secondaire local pour une table à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Créez une instance de la classe `DescribeTableRequest` pour fournir l'information de requête. Vous devez fournir le nom de la table.
3. Exécutez la méthode `describeTable` en fournissant l'objet de demande comme paramètre.
4.

L'exemple de code C# suivant présente les étapes précédentes.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
    response.DescribeTableResult.Table.LocalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");

    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
```

```
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

Interroger un index secondaire local

Vous pouvez utiliser l'opération `Query` sur un index secondaire local de la même manière que vous utilisez l'opération `Query` sur une table. Vous devez spécifier le nom d'index, les critères de requête pour la clé de tri d'index et les attributs que vous souhaitez renvoyer. Dans cet exemple, l'index est `AlbumTitleIndex` et la clé de tri d'index est `AlbumTitle`.

Les seuls attributs renvoyés sont ceux qui ont été projetés dans l'index. Vous pourriez également modifier cette requête pour sélectionner des attributs autres que de clé, mais cela nécessiterait une activité d'extraction de table relativement coûteuse. Pour plus d'informations sur les extractions de table, consultez Projections d’attribut

Voici les étapes à suivre pour interroger un index secondaire local à l'aide de l'API de bas niveau .NET.

1. Créez une instance de la classe `AmazonDynamoDBClient`.
2. Créez une instance de la classe `QueryRequest` pour fournir l'information de requête.
3. Exécutez la méthode `query` en fournissant l'objet de demande comme paramètre.

L'exemple de code C# suivant présente les étapes précédentes.

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

Exemple : index secondaires locaux utilisant l'API de bas niveau AWS SDK pour .NET

L'exemple de code C# suivant montre comment utiliser les index secondaires locaux dans Amazon DynamoDB. L'exemple crée une table nommée `CustomerOrders` avec une clé de partition `CustomerId` et une clé de tri `OrderId`. Il y a deux index secondaires locaux sur cette table :

- `OrderCreationDateIndex` – La clé de tri est.`OrderCreationDate`, et les attributs suivants sont projetés dans l'index :
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`

- `IsOpenIndex` – La clé de tri est `IsOpen` et tous les attributs de table sont projetés dans l'index.

Une fois la table `CustomerOrders` créée, le programme charge la table avec des données représentant des commandes de clients. Il interroge ensuite les données à l'aide d'index secondaires locaux. Enfin, le programme supprime la table `CustomerOrders`.

Pour obtenir des instructions pas à pas afin de tester l'exemple suivant, consultez Exemples de code .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
                LoadData();

                Query(null);
                Query("IsOpenIndex");
                Query("OrderCreationDateIndex");

                DeleteTable(tableName);

                Console.WriteLine("To continue, press Enter");
```

```
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void CreateTable()
    {
        var createTableRequest =
            new CreateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput =
                new ProvisionedThroughput()
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                }
            };

        var attributeDefinitions = new List<AttributeDefinition>()
    {
        // Attribute definitions for table primary key
        { new AttributeDefinition() {
              AttributeName = "CustomerId", AttributeType = "S"
          } },
        { new AttributeDefinition() {
              AttributeName = "OrderId", AttributeType = "N"
          } },
        // Attribute definitions for index primary key
        { new AttributeDefinition() {
              AttributeName = "OrderCreationDate", AttributeType = "N"
          } },
        { new AttributeDefinition() {
              AttributeName = "IsOpen", AttributeType = "N"
          }}
    };

        createTableRequest.AttributeDefinitions = attributeDefinitions;

        // Key schema for table
        var tableKeySchema = new List<KeySchemaElement>()
    {
```

```
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderId", KeyType = "RANGE"
              } }                                                   //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                      //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderCreationDate", KeyType = "RANGE"
              } }                                                                 //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
                ProjectionType = "INCLUDE"
            };

            var nonKeyAttributes = new List<string>()
        {
            "ProductCategory",
            "ProductName"
        };
            projection.NonKeyAttributes = nonKeyAttributes;
```

```
            orderCreationDateIndex.Projection = projection;

            localSecondaryIndexes.Add(orderCreationDateIndex);

            // IsOpenIndex
            LocalSecondaryIndex isOpenIndex
                = new LocalSecondaryIndex()
                {
                    IndexName = "IsOpenIndex"
                };

            // Key schema for IsOpenIndex
            indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              }},                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "IsOpen", KeyType = "RANGE"
              }}                                                  //Sort key
        };

            // Projection (all attributes) for IsOpenIndex
            projection = new Projection()
            {
                ProjectionType = "ALL"
            };

            isOpenIndex.KeySchema = indexKeySchema;
            isOpenIndex.Projection = projection;

            localSecondaryIndexes.Add(isOpenIndex);

            // Add index definitions to CreateTable request
            createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {
```

```
Console.WriteLine("\n*********************************************************\n");
           Console.WriteLine("Querying table " + tableName + "...");

           QueryRequest queryRequest = new QueryRequest()
           {
               TableName = tableName,
               ConsistentRead = true,
               ScanIndexForward = true,
               ReturnConsumedCapacity = "TOTAL"
           };


           String keyConditionExpression = "CustomerId = :v_customerId";
           Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
           {":v_customerId", new AttributeValue {
                S = "bob@example.com"
            }}
       };


           if (indexName == "IsOpenIndex")
           {
               Console.WriteLine("\nUsing index: '" + indexName
                         + "': Bob's orders that are open.");
               Console.WriteLine("Only a user-specified list of attributes are
returned\n");
               queryRequest.IndexName = indexName;

               keyConditionExpression += " and IsOpen = :v_isOpen";
               expressionAttributeValues.Add(":v_isOpen", new AttributeValue
               {
                   N = "1"
               });

               // ProjectionExpression
               queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
           }
           else if (indexName == "OrderCreationDateIndex")
           {
               Console.WriteLine("\nUsing index: '" + indexName
                         + "': Bob's orders that were placed after 01/31/2013.");
```

```
                Console.WriteLine("Only the projected attributes are returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and OrderCreationDate > :v_Date";
                expressionAttributeValues.Add(":v_Date", new AttributeValue
                {
                    N = "20130131"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
            }
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "OrderId" || attr == "IsOpen"
                        || attr == "OrderCreationDate")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
            }
            Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
```

```
            client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        public static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130101"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "The Great Outdoors"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            PutItemRequest putItemRequest = new PutItemRequest
```

```
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Bike"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Super Mountain"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "ORDER RECEIVED"
            };
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);
```

```
item = new Dictionary<string, AttributeValue>();
item["CustomerId"] = new AttributeValue
{
    S = "alice@example.com"
};
item["OrderId"] = new AttributeValue
{
    N = "3"
};
/* no IsOpen attribute */
item["OrderCreationDate"] = new AttributeValue
{
    N = "20130304"
};
item["ProductCategory"] = new AttributeValue
{
    S = "Music"
};
item["ProductName"] = new AttributeValue
{
    S = "A Quiet Interlude"
};
item["OrderStatus"] = new AttributeValue
{
    S = "IN TRANSIT"
};
item["ShipmentTrackingId"] = new AttributeValue
{
    S = "176493"
};
putItemRequest = new PutItemRequest
{
    TableName = tableName,
    Item = item,
    ReturnItemCollectionMetrics = "SIZE"
};
client.PutItem(putItemRequest);

item = new Dictionary<string, AttributeValue>();
item["CustomerId"] = new AttributeValue
{
    S = "bob@example.com"
};
```

```
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130111"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Movie"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Calm Before The Storm"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "SHIPPING DELAY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "859323"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
```

```
            {
                N = "20130124"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "E-Z Listening"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "756943"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
```

```
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Symphony 9"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "645193"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "4"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130222"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Hardware"
            };
            item["ProductName"] = new AttributeValue
            {
```

```
            S = "Extra Heavy Hammer"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "PACKING ITEMS"
        };
        /* no ShipmentTrackingId attribute */
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "5"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130309"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Book"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "How To Cook"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "IN TRANSIT"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "440185"
```

```
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "6"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130318"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Luggage"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Really Big Suitcase"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "893927"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
```

```
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "7"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130324"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Golf"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "PGA Pro II"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "OUT FOR DELIVERY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "383283"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);
        }

        private static void WaitUntilTableReady(string tableName)
        {
```

```
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
```

```
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Utilisation d'index secondaires locaux dans la AWS CLI de DynamoDB

Vous pouvez utiliser AWS CLI pour créer une table Amazon DynamoDB avec un ou plusieurs index secondaires locaux, décrire les index sur la table, et effectuer des requêtes à l'aide des index.

Rubriques

- Créer une table avec un index secondaire local
- Décrire une table avec un index secondaire local
- Interroger un index secondaire local

Créer une table avec un index secondaire local

Vous devez créer des index secondaires locaux au moment où vous créez une table. Pour ce faire, utilisez le paramètre `create-table` et fournissez vos spécifications pour un ou plusieurs index secondaires locaux. L'exemple suivant crée une table (`Music`) destinée à accueillir des données informations sur des chansons dans une collection musicale. La clé de partition est `Artist`, et la clé de tri `SongTitle`. Un index secondaire, `AlbumTitleIndex`, sur l'attribut `AlbumTitle` facilite les requêtes par titre d'album.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                      {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
```

```
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

Vous devez attendre que DynamoDB crée la table et définisse l'état de celle-ci sur `ACTIVE`. Après cela, vous pouvez commencer à insérer des éléments de données dans la table. Vous pouvez utiliser la commande describe-table pour déterminer l'état de la création de table.

Décrire une table avec un index secondaire local

Pour obtenir des informations concernant les index secondaires locaux sur une table, utilisez le paramètre `describe-table`. Pour chaque index, vous pouvez accéder à son nom, à son schéma de clé et aux attributs projetés.

```
aws dynamodb describe-table --table-name Music
```

Interroger un index secondaire local

Vous pouvez utiliser l'opération `query` sur un index secondaire local de la même manière que vous utilisez l'opération `query` sur une table. Vous devez spécifier le nom d'index, les critères de requête pour la clé de tri d'index et les attributs que vous souhaitez renvoyer. Dans cet exemple, l'index est `AlbumTitleIndex` et la clé de tri d'index est `AlbumTitle`.

Les seuls attributs renvoyés sont ceux qui ont été projetés dans l'index. Vous pourriez également modifier cette requête pour sélectionner des attributs autres que de clé, mais cela nécessiterait une activité d'extraction de table relativement coûteuse. Pour plus d'informations sur les extractions de table, consultez Projections d'attribut.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

Gestion des flux de travail complexes avec des transactions DynamoDB

Les transactions Amazon DynamoDB simplifient l'expérience des développeurs en apportant des modifications coordonnées all-or-nothing à plusieurs éléments à la fois au sein des tables et entre

elles. Les transactions introduisent l'atomicité, la cohérence, l'isolation et la durabilité (ACID) dans DynamoDB, ce qui permet de maintenir facilement l'exactitude des données dans vos applications.

Vous pouvez utiliser la lecture et l' APIs écriture transactionnelles DynamoDB pour gérer des flux de travail métier complexes qui nécessitent l'ajout, la mise à jour ou la suppression de plusieurs éléments en une seule opération. all-or-nothing Par exemple, un développeur de jeux vidéo peut ainsi s'assurer que les profils des joueurs sont mis à jour correctement lorsqu'ils échangent des objets ou effectuent des achats dans un jeu.

Avec l'API d'écriture de transaction, vous pouvez regrouper plusieurs actions `Put`, `Update`, `Delete` et `ConditionCheck`. Ensuite, vous pouvez soumettre les actions comme une seule opération `TransactWriteItems` qui réussit ou échoue en tant qu'unité. Il en va de même avec les actions `Get`, que vous pouvez regrouper et soumettre en une seule opération `TransactGetItems`.

L'activation des transactions pour vos tables DynamoDB n'occasionne pas de frais supplémentaires . Vous ne payez que pour les lectures ou écritures qui font partie de votre transaction. DynamoDB effectue deux lectures ou écritures sous-jacentes de chaque élément faisant partie de la transaction : l'une pour préparer la transaction, l'autre pour la valider. Ces deux read/write opérations sous-jacentes sont visibles dans vos CloudWatch statistiques Amazon.

Pour commencer à utiliser les transactions DynamoDB, téléchargez le AWS dernier SDK ou le ().AWS Command Line InterfaceAWS CLI Suivez ensuite l'Exemple de transactions DynamoDB.

Les sections suivantes fournissent un aperçu détaillé de la transaction APIs et de la manière dont vous pouvez les utiliser dans DynamoDB.

Rubriques

- Fonctionnement de transactions Amazon DynamoDB
- Utilisation d'IAM avec des transactions DynamoDB
- Exemple de transactions DynamoDB

Fonctionnement de transactions Amazon DynamoDB

Avec les transactions Amazon DynamoDB, vous pouvez regrouper plusieurs actions et les soumettre en une all-or-nothing `TransactWriteItems` seule opération. `TransactGetItems` Les sections suivantes décrivent les opérations d'API, la gestion de la capacité, les bonnes pratiques et d'autres détails concernant l'utilisation des opérations transactionnelles dans DynamoDB.

Rubriques

- TransactWriteItems API
- TransactGetItems API
- Niveaux d’isolement pour les transactions DynamoDB
- Gestion des conflits de transactions dans DynamoDB
- Utilisation du mode transactionnel APIs dans DynamoDB Accelerator (DAX)
- Gestion de capacité pour les transactions
- Bonnes pratiques en matière de transactions
- Utilisation du mode transactionnel APIs avec des tables globales
- Transactions DynamoDB et bibliothèque cliente de transactions AWSLabs

TransactWriteItems API

`TransactWriteItems`est une opération d'écriture synchrone et idempotente qui regroupe jusqu'à 100 actions d'écriture en une seule opération. all-or-nothing Ces actions peuvent cibler jusqu'à 100 éléments distincts dans une ou plusieurs tables DynamoDB au sein du AWS même compte et de la même région. La taille d’agrégation des éléments dans la transaction ne peut pas dépasser 4 Mo. Les actions sont exécutées de manière atomique, de sorte qu’elles réussissent toutes ou aucune ne réussit.

> **Note**
>
> - Une opération `TransactWriteItems` diffère d’une opération `BatchWriteItem` en ceci que toutes les actions qui y sont contenues doivent réussir, faute de quoi aucune modification du tout n’est apportée. Avec une opération `BatchWriteItem`, il est possible que certaines actions du lot seulement réussissent, alors que d’autres échouent.
> - Les transactions ne peuvent pas être effectuées au moyen d’index

Vous ne pouvez pas cibler le même élément avec plusieurs opérations contenues dans la même transaction. Par exemple, vous ne pouvez pas effectuer un `ConditionCheck` et une action `Update` sur le même élément de la même transaction.

Vous pouvez ajouter les types d’actions suivants à une transaction :

- `Put` – Lance une opération `PutItem` pour créer un élément ou remplacer un ancien élément par un nouveau, de manière conditionnelle ou sans spécifier de condition.
- `Update` – Lance une opération `UpdateItem` pour modifier les attributs d'un élément existant, ou ajouter un élément à la table s'il n'existe pas. Utilisez cette action pour ajouter, supprimer ou mettre à jour des attributs d'un élément existant, avec ou sans conditions.
- `Delete` – Lance une opération `DeleteItem` pour supprimer un élément unique d'une table identifiée par sa clé primaire.
- `ConditionCheck` – Vérifie l'existence d'un élément ou l'état d'attributs spécifiques de celui-ci.

Lorsqu'une transaction est terminée dans DynamoDB, ses modifications commencent à se propager aux index secondaires globaux GSIs (), aux flux et aux sauvegardes. Cette propagation se produit progressivement : les enregistrements de flux d'une même transaction peuvent s'afficher à des moments différents et s'entrelacer avec des enregistrements d'autres transactions. Les consommateurs de flux ne doivent pas présumer de l'atomicité des transactions ou des garanties d'ordre.

Pour garantir un instantané atomique des éléments modifiés lors d'une transaction, utilisez l' TransactGetItems opération pour lire tous les éléments pertinents ensemble. Cette opération fournit une vue cohérente des données, vous permettant soit de voir toutes les modifications apportées à une transaction terminée, soit de n'en voir aucune.

La propagation n'étant pas immédiate, si une table est restaurée à partir de backup (RestoreTableFromBackup) ou exportée à un point dans le temps (ExportTableToPointInTime) en cours de propagation, elle ne peut contenir que certaines des modifications apportées lors d'une transaction récente.

Idempotence

En option, vous pouvez inclure un jeton client lorsque vous faites un appel `TransactWriteItems` afin de garantir l'idempotence de la demande. Des transactions idempotentes évitent des erreurs d'application si la même opération est soumise plusieurs fois suite à une expiration de la connexion ou à tout autre problème de connectivité.

Si l'appel initial de `TransactWriteItems` réussit, les appels suivants de `TransactWriteItems` avec le même jeton client aboutissent sans apporter aucune modification. Si le paramètre `ReturnConsumedCapacity` est défini, l'appel `TransactWriteItems` initial renvoie le nombre d'unités de capacité en écriture consommées par l'exécution de ces modifications.

Les appels `TransactWriteItems` subséquents avec le même jeton client renvoient le nombre d'unités de capacité en lecture consommées par la lecture de l'élément.

Considérations importantes concernant l'idempotence

- Un jeton client est valide pendant 10 minutes après la fin de la demande qui l'utilise. Après 10 minutes, toute demande utilisant le même jeton client est traitée comme une nouvelle demande. Évitez donc de réutiliser le même jeton client pour la même demande après 10 minutes.
- Si vous répétez une demande avec le même jeton client dans la fenêtre d'idempotence de 10 minutes, mais modifiez un autre paramètre de demande, DynamoDB renvoie une exception `IdempotentParameterMismatch`.

Gestion des erreurs d'écriture

Les transactions d'écriture échouent dans les conditions suivantes :

- Lorsqu'une condition d'une des expressions de condition n'est pas remplie.
- Lorsqu'une erreur de validation de la transaction se produit parce que plusieurs actions de la même opération `TransactWriteItems` ciblent le même élément.
- Lorsqu'une demande `TransactWriteItems` est en conflit avec une opération `TransactWriteItems` en cours sur un ou plusieurs éléments dans la demande `TransactWriteItems`. Dans ce cas, la demande échoue avec une exception `TransactionCanceledException`.
- Lorsque la capacité allouée est insuffisante pour achever la transaction.
- Lorsqu'un élément devient trop volumineux (plus de 400 ko), qu'un index secondaire local (LSI) devient trop volumineux ou qu'une erreur de validation similaire se produit en raison de modifications effectuées par la transaction.
- En cas d'erreur de la part de l'utilisateur, par exemple un format de données incorrect.

Pour en savoir plus sur la gestion des conflits avec les opérations `TransactWriteItems`, consultez Gestion des conflits de transactions dans DynamoDB.

TransactGetItems API

`TransactGetItems` est une opération de lecture synchrone qui regroupe jusqu'à 100 actions `Get`. Ces actions peuvent cibler jusqu'à 100 éléments distincts dans une ou plusieurs tables DynamoDB

au sein d'un même compte et d'une AWS même région. La taille d’agrégation des éléments dans la transaction ne peut pas dépasser 4 Mo.

Les actions `Get` sont exécutées de manière atomique, de sorte qu’elles réussissent toutes ou échouent toutes.

- `Get` – Lance une opération `GetItem` afin d’extraire un ensemble d’attributs pour l’élément avec la clé primaire donnée. Si aucun élément correspondant n’est trouvé, `Get` ne renvoie pas de données.

Gestion des erreurs de lecture

Les transactions de lecture échouent dans les conditions suivantes :

- Lorsqu’une demande `TransactGetItems` est en conflit avec une opération `TransactWriteItems` en cours sur un ou plusieurs éléments dans la demande `TransactGetItems`. Dans ce cas, la demande échoue avec une exception `TransactionCanceledException`.
- Lorsque la capacité allouée est insuffisante pour achever la transaction.
- En cas d’erreur de la part de l’utilisateur, par exemple un format de données incorrect.

Pour en savoir plus sur la gestion des conflits avec les opérations `TransactGetItems`, consultez Gestion des conflits de transactions dans DynamoDB.

Niveaux d’isolement pour les transactions DynamoDB

Les niveaux d’isolement des opérations transactionnelles (`TransactWriteItems` ou `TransactGetItems`) et autres opérations sont les suivants.

SERIALIZABLE

Un isolement sérialisable assure que les résultats de plusieurs opérations concourantes sont les mêmes que si aucune opération ne commençait avant que la précédente soit terminée.

Il y a un isolement sérialisable entre les types d’opérations suivants :

- Entre toute opération transactionnelle et toute opération d’écriture standard (`PutItem`, `UpdateItem` ou `DeleteItem`).
- Entre toute opération transactionnelle et toute opération de lecture standard (`GetItem`).

- Entre une opération `TransactWriteItems` et une opération `TransactGetItems`.

Bien qu'il y ait un isolement sérialisable entre les opérations transactionnelles et chaque écriture individuelle d'une opération `BatchWriteItem`, il n'y en a pas entre la transaction et l'opération `BatchWriteItem` en tant qu'ensemble.

De même, le niveau d'isolation entre une opération transactionnelle et un `GetItems` dans une opération `BatchGetItem` est sérialisable. Mais le niveau d'isolation entre la transaction et l'opération `BatchGetItem` en tant qu'unité est read-committed.

Une demande `GetItem` unique est sérialisable par rapport à une demande `TransactWriteItems` de deux façons, avant ou après la demande `TransactWriteItems`. Plusieurs demandes `GetItem` par rapport à des clés dans une demande `TransactWriteItems` simultanée peuvent s'exécuter dans n'importe quel ordre. C'est pourquoi les résultats sont validés en lecture.

Par exemple, si des demandes `GetItem` pour des éléments A et B sont exécutées en même temps qu'une demande `TransactWriteItems` qui modifie les éléments A et B, il y a quatre possibilités :

- Les deux demandes `GetItem` sont exécutées avant la demande `TransactWriteItems`.
- Les deux demandes `GetItem` sont exécutées après la demande `TransactWriteItems`.
- La demande `GetItem` pour l'élément A est exécutée avant la demande `TransactWriteItems`. Pour l'élément B, la demande `GetItem` est exécutée après la demande `TransactWriteItems`.
- La demande `GetItem` pour l'élément B est exécutée avant la demande `TransactWriteItems`. Pour l'élément A, la demande `GetItem` est exécutée après la demande `TransactWriteItems`.

Vous devez utiliser la demande `TransactGetItems` si vous préférez un niveau d'isolement sérialisable pour plusieurs demandes `GetItem`.

Si une lecture non transactionnelle est effectuée sur plusieurs éléments faisant partie de la même demande d'écriture de transaction en cours, il est possible que vous puissiez lire le nouvel état de certains éléments et l'ancien état des autres éléments. Vous pouvez lire le nouvel état de tous les éléments qui faisaient partie de la demande d'écriture de transaction seulement en cas de réception d'une réponse de réussite de l'écriture de transaction indiquant que la transaction est terminée.

Une fois la transaction effectuée et la réponse reçue, les opérations ultérieures de lecture cohérente à terme peuvent toujours renvoyer l'ancien état pendant une courte période en raison du modèle de cohérence à terme de DynamoDB. Pour garantir la lecture du plus grand up-to-date nombre de

données immédiatement après une transaction, vous devez utiliser des lectures très cohérentes en définissant le paramètre `ConsistentRead` sur true.

READ-COMMITTED

L'isolement validé en lecture garantit que les opérations de lecture renvoient toujours des valeurs validées pour un élément. La lecture ne présentera jamais une vue de l'élément représentant un état issu d'une écriture transactionnelle qui n'a pas abouti. Un isolement validé en lecture n'empêche pas des modifications de l'élément juste après la lecture.

Le niveau d'isolement est validé en lecture entre toute opération transactionnelle et toute opération de lecture impliquant plusieurs lectures standard (`BatchGetItem`, `Query` ou `Scan`). Si une écriture transactionnelle met à jour un élément au milieu d'une opération `BatchGetItem`, `Query` ou `Scan`, la partie suivante de l'opération de lecture renvoie la valeur nouvellement validée (avec `ConsistentRead`) ou éventuellement une valeur déjà validée (lectures éventuellement cohérentes).

Résumé des opérations

Pour résumer, le tableau qui suit montre les niveaux d'isolement entre une opération transactionnelle (`TransactWriteItems` ou `TransactGetItems`) et d'autres opérations :

Opération	Niveau d'isolement
`DeleteItem`	Sérialisable
`PutItem`	Sérialisable
`UpdateItem`	Sérialisable
`GetItem`	Sérialisable
`BatchGetItem`	Validé en lecture*
`BatchWriteItem`	NON sérialisable*
`Query`	Validé en lecture
`Scan`	Validé en lecture
Autre opération transactionnelle	Sérialisable

Les niveaux marqués d'un astérisque (*) s'appliquent à l'opération dans son ensemble. Toutefois, les actions individuelles au sein de ces opérations ont un niveau d'isolement sérialisable.

Gestion des conflits de transactions dans DynamoDB

Un conflit transactionnel peut se produire lors de demandes simultanées au niveau de l'élément sur un élément au sein d'une transaction. Les conflits de transaction peuvent se produire dans les scénarios suivants :

- Une requête `PutItem`, `UpdateItem` ou `DeleteItem` pour un élément est en conflit avec une requête `TransactWriteItems` en cours qui inclut le même élément.
- Un élément au sein d'une requête `TransactWriteItems` fait partie d'une autre requête `TransactWriteItems` en cours.
- Un élément au sein d'une requête `TransactGetItems` fait partie d'une autre requête `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem` ou `DeleteItem` en cours.

> **Note**
>
> - Quand une requête `PutItem`, `UpdateItem` ou `DeleteItem` est rejetée, la requête échoue avec un code `TransactionConflictException`.
> - Si toute requête au niveau de l'élément au sein de `TransactWriteItems` ou `TransactGetItems` est rejetée, la requête échoue avec un code `TransactionCanceledException`. Si cette demande échoue,AWS SDKs ne réessayez pas.
>
> Si vous utilisez le AWS SDK pour Java, l'exception contient la liste de CancellationReasons, ordonnée en fonction de la liste des éléments du paramètre de `TransactItems` demande. Pour d'autres langues, une représentation sous forme de chaîne de la liste est incluse dans le message d'erreur de l'exception.
> - Si une opération `TransactWriteItems` en cours ou une opération `TransactGetItems` est en conflit avec une demande `GetItem` concourante, les deux opérations peuvent réussir.

La TransactionConflict CloudWatch métrique est incrémentée pour chaque demande échouée au niveau de l'élément.

Utilisation du mode transactionnel APIs dans DynamoDB Accelerator (DAX)

Les demandes `TransactWriteItems` et `TransactGetItems` sont toutes deux prises en charge dans DynamoDB Accelerator (DAX) avec les mêmes niveaux d'isolement que dans DynamoDB.

La demande `TransactWriteItems` écrit via DAX. DAX transmet un appel `TransactWriteItems` à DynamoDB et renvoie la réponse. Pour remplir le cache après l'écriture, DAX appelle `TransactGetItems` en arrière-plan pour chaque élément dans l'opération `TransactWriteItems`, qui consomme des unités de capacité de lecture supplémentaires. (Pour en savoir plus, consultez Gestion de capacité pour les transactions.) Cette fonctionnalité vous permet de simplifier la logique de votre application et d'utiliser DAX pour les opérations transactionnelles et non transactionnelles.

Les appels `TransactGetItems` sont transmis via DAX sans les éléments mis en cache localement. Il s'agit du même comportement que pour une lecture très cohérente APIs dans DAX.

Gestion de capacité pour les transactions

L'activation des transactions pour vos tables DynamoDB n'occasionne pas de frais supplémentaires . Vous ne payez que pour les lectures ou écritures qui font partie de votre transaction. DynamoDB effectue deux lectures ou écritures sous-jacentes de chaque élément faisant partie de la transaction : l'une pour préparer la transaction, l'autre pour la valider. Les deux read/write opérations sous-jacentes sont visibles dans vos CloudWatch statistiques Amazon.

Planifiez les lectures et écritures supplémentaires requises par le protocole transactionnel APIs lorsque vous allouez de la capacité à vos tables. Par exemple, supposons que votre application exécute une transaction par seconde et que chaque transaction écrit trois éléments de 500 octets dans votre table. Chaque article nécessite deux unités de capacité d'écriture (WCUs) : une pour préparer la transaction et une pour valider la transaction. Par conséquent, vous devez en fournir six WCUs à la table.

Si vous utilisiez DynamoDB Accelerator (DAX) dans l'exemple précédent, vous utiliseriez également deux unités de capacité de lecture RCUs () pour chaque élément de l'appel. `TransactWriteItems` Vous devrez donc en prévoir six supplémentaires RCUs à la table.

De même, si votre application exécute une transaction de lecture par seconde et que chaque transaction lit trois éléments de 500 octets dans votre table, vous devez fournir six unités de capacité de lecture (RCUs) à la table. La lecture de chaque élément en nécessite deux RCUs : un pour préparer la transaction et un pour valider la transaction.

En outre, le comportement par défaut du SDK est de ressayer les transactions en cas d'exception `TransactionInProgressException`. Prévoyez les unités de capacité de lecture supplémentaires (RCUs) consommées par ces nouvelles tentatives. Il en va de même si vous réessayez des transactions dans votre propre code à l'aide d'un `ClientRequestToken`.

Bonnes pratiques en matière de transactions

Lorsque vous utilisez des transactions DynamoDB, prenez en compte les bonnes pratiques recommandées suivantes.

- Activez la scalabilité automatique sur vos tables ou veillez à allouer suffisamment de capacité de débit pour effectuer les deux opérations de lecture et écriture pour chaque élément de votre transaction.
- Si vous n'utilisez pas le SDK AWS fourni, incluez un `ClientRequestToken` attribut lorsque vous effectuez un `TransactWriteItems` appel pour vous assurer que la demande est idempotente.
- Ne regroupez pas d'opérations en une transaction si ce n'est pas nécessaire. Par exemple, si une transaction unique avec 10 opérations peut être subdivisée en plusieurs transactions sans compromettre l'exactitude de l'application, nous recommandons de scinder la transaction. Des transactions plus simples améliorent le débit et sont plus susceptibles de réussir.
- Si plusieurs transactions mettent à jour simultanément les mêmes éléments, il peut y avoir des conflits qui annulent ces transactions. Nous recommandons de suivre les bonnes pratiques DynamoDB en matière de modélisation des données pour minimiser de tels conflits.
- Si un ensemble d'attributs est souvent mis à jour sur plusieurs éléments dans le cadre d'une seule et même transaction, envisagez de regrouper les attributs dans un seul élément afin de réduire la portée de la transaction.
- Évitez d'utiliser des transactions pour ingérer des données en masse. Pour les écritures en masse, il est préférable d'utiliser `BatchWriteItem`.

Utilisation du mode transactionnel APIs avec des tables globales

Les opérations transactionnelles fournissent des garanties d'atomicité, de cohérence, d'isolation et de durabilité (ACID) uniquement dans la AWS région où l'API d'écriture a été invoquée. Les transactions des tables globales ne sont pas prises en charge dans toutes les régions. Par exemple, supposons que vous disposiez d'une table globale avec des réplicas dans les régions USA Est (Ohio) et USA Ouest (Oregon) et que vous effectuiez une opération `TransactWriteItems` dans la région USA Est (Virginie du Nord). Vous risquez d'observer des transactions partiellement terminées dans la

région USA Ouest (Oregon) lorsque les modifications sont répliquées. Les modifications ne sont répliquées dans les autres régions qu'une fois validées dans la région source.

Transactions DynamoDB et bibliothèque cliente de transactions AWSLabs

Les transactions DynamoDB constituent un remplacement plus rentable, plus robuste et plus performant de AWSLabsla bibliothèque cliente de transactions. Nous vous suggérons de mettre à jour vos applications pour utiliser la transaction native côté serveur. APIs

Utilisation d'IAM avec des transactions DynamoDB

Vous pouvez utiliser Gestion des identités et des accès AWS(IAM) pour restreindre les actions que les opérations transactionnelles peuvent effectuer dans Amazon DynamoDB. Pour en savoir plus sur l'utilisation de politiques IAM dans DynamoDB, consultez Politiques basées sur l'identité pour DynamoDB.

Les autorisations pour `Put`, `Update`, `Delete` et `Get` sont régies par celle utilisées pour les opérations `PutItem`, `UpdateItem`, `DeleteItem` et `GetItem` sous-jacentes. Pour l'action `ConditionCheck`, vous pouvez utiliser l'autorisation `dynamodb:ConditionCheckItem` dans des politiques IAM.

Vous trouverez ci-dessous des exemples de politiques IAM que vous pouvez utiliser pour configurer des transactions DynamoDB.

Exemple 1 : autoriser les opérations transactionnelles

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
```

```
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

Exemple 2 : autoriser uniquement les opérations transactionnelles

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

Exemple 3 : autoriser les lectures et écritures non transactionnelles, bloquer les lectures et écritures transactionnelles

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
```

```
}
```

Exemple 4 : Empêcher le renvoi d'informations en ConditionCheck cas de panne

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

Exemple de transactions DynamoDB

À titre d’exemple de situation dans laquelle des transactions Amazon DynamoDB peuvent être utiles, considérez cette application Java pour une place de marché en ligne.

L’application dispose de trois tables DynamoDB dans le backend :

- `Customers` – Cette table stocke des détails sur les clients de la place de marché. Sa clé primaire est un identifiant unique `CustomerId`.
- `ProductCatalog` – Ce tableau stocke des détails tels que le prix et la disponibilité des produits en vente sur la place de marché. Sa clé primaire est un identifiant unique `ProductId`.

- `Orders` – Cette table stocke des détails sur les commandes en provenance de la place de marché. Sa clé primaire est un identifiant unique `OrderId`.

Passation d'une commande

Les extraits de code suivants montrent comment utiliser des transactions DynamoDB pour coordonner les différentes étapes de la création et du traitement d'une commande. L'utilisation d'une seule all-or-nothing opération garantit qu'en cas d'échec d'une partie de la transaction, aucune action de la transaction n'est exécutée et aucune modification n'est apportée.

Dans cet exemple, vous configurez une commande à partir d'un client dont le `customerId` est `09e8e9c8-ec48`. Vous l'exécutez ensuite en tant que transaction unique à l'aide du simple flux de traitement de commandes suivant :

1. Déterminez si l'ID client est valide.
2. Assurez-vous que l'état du produit est `IN_STOCK` et mettez à jour son état en `SOLD`.
3. Assurez-vous que la commande n'existe pas déjà, puis créez la commande.

Valider le client

Commencez par définir une action pour vérifier qu'un client avec `customerId` égal à `09e8e9c8-ec48` existe dans la table client.

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

Mettre à jour l'état du produit

Ensuite, définissez une action pour mettre à jour l'état du produit sur `SOLD` si la condition que l'état du produit soit actuellement défini sur `IN_STOCK` est `true`. La définition du paramètre

`ReturnValuesOnConditionCheckFailure` a pour effet de renvoyer l'article si l'attribut d'état de produit de celui-ci n'est pas `IN_STOCK`.

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));

Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

Créer la commande

Enfin, créez la commande pour autant qu'il n'existe pas de commande avec cet `OrderId`.

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

Exécuter la transaction

L'exemple suivant montre comment exécuter les actions définies précédemment en une seule all-or-nothing opération.

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
    }
```

Lecture des détails de la commande

L'exemple suivant montre comment lire la commande terminée de manière transactionnelle dans les tables `Orders` et `ProductCatalog`.

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
Get readCreatedOrder = new Get()
```

```
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

Récupération de données de modification avec Amazon DynamoDB

Bon nombre d'applications bénéficient de la récupération des modifications apportées aux éléments stockés dans une table DynamoDB au moment elles se produisent. Voici quelques exemples de cas d'utilisation.

- Une application mobile populaire modifie des données dans une table DynamoDB au rythme de milliers de mises à jour par seconde. Une autre application capture et stocke les données relatives à ces mises à jour, fournissant des mesures near-real-time d'utilisation pour l'application mobile.
- Une application financière modifie des données boursières dans une table DynamoDB. Différentes applications exécutées en parallèle suivent ces changements en temps réel value-at-risk, calculent et rééquilibrent automatiquement les portefeuilles en fonction de l'évolution du cours des actions.
- Des capteurs dans des véhicules de transport et des équipements industriels envoient des données à une table DynamoDB. Différentes applications contrôlent les performances et envoient

des alertes par messagerie lors de la détection d'un problème, prédisent des défaillances potentielles en appliquant des algorithmes de machine learning, et compressent et archivent les données dans Amazon Simple Storage Service (Amazon S3).

- Une application envoie automatiquement des notifications sur les appareils mobiles de tous les amis dans un groupe dès qu'un ami télécharge une nouvelle image.
- Un nouveau client ajoute des données à une table DynamoDB. Cet événement appelle une autre application qui envoie un e-mail de bienvenue au nouveau client.

DynamoDB prend en charge le streaming des enregistrements de récupération de données de modification au niveau élément en quasi-temps réel. Vous pouvez créer des applications qui consomment ces flux et entreprennent des actions basées sur le contenu.

> **Note**
>
> L'ajout de balises à DynamoDB Streams et l'utilisation du contrôle d'accès par attributs (ABAC) avec DynamoDB Streams ne sont pas pris en charge.

La vidéo suivante vous présente le concept de capture des données de modification.

Modes de capacité table

Rubriques

- Options de streaming pour la récupération de données de modification
- Utilisation de Kinesis Data Streams pour récupérer les modifications apportées à DynamoDB
- Modifier la récupération de données pour DynamoDB Streams

Options de streaming pour la récupération de données de modification

DynamoDB offre deux modèles de streaming pour la récupération de données de modification : Kinesis Data Streams pour DynamoDB et DynamoDB Streams.

Afin de vous aider à choisir la solution appropriée pour votre application, le tableau suivant récapitule les fonctions de chaque modèle de streaming.

Propriétés	Kinesis Data Streams pour DynamoDB	DynamoDB Streams
Conservation des données	Jusqu'à 1 an.	24 heures
Prise en charge de la bibliothèque client Kinesis (Kinesis Client Library, KCL)	Prend en charge KCL versions 1.X, 2.X et 3.X.	Prend en charge KCL versions 1.X et 2.X.
Nombre de consommateurs	Jusqu'à 5 consommateurs simultanés par partition, ou 20 avec la diffusion améliorée.	Jusqu'à 2 consommateurs simultanés par partition.
Quotas de débit	Illimité.	Soumis à des quotas de débit par table DynamoDB et par région.AWS
Modèle de livraison de l'enregistrement	Transférez le modèle via HTTP en utilisant GetRecords grâce à un ventilateur amélioré, Kinesis Data Streams transmet les enregistrements via HTTP/2 en utilisant . SubscribeToShard	Extraire le modèle via HTTP en utilisant GetRecords.
Classement des enregistrements	L'attribut d'horodatage de chaque enregistrement de flux permet d'identifier l'ordre réel dans lequel les modifications se sont produites dans la table DynamoDB.	Pour chaque élément modifié dans une table DynamoDB, les enregistrements de flux apparaissent dans l'ordre des modifications réelles.
Enregistrements en double	Des enregistrements en double peuvent apparaître dans le flux.	Aucun enregistrement en double n'apparaît dans le flux.
Options de traitement de flux	Traitez les enregistrements de flux à l'aide d'AWS Lambda,	Traitez les enregistrements de flux à l'aide de AWS Lambda

Propriétés	Kinesis Data Streams pour DynamoDB	DynamoDB Streams
	du service géré Amazon pour Apache Flink, de Kinesis Data Firehose ou des tâches ETL en streaming d'AWS Glue.	ou de l'Adaptateur DynamoDB Streams Kinesis.
Niveau de durabilité	Zones de disponibilité pour assurer un basculement automatique sans interruption.	Zones de disponibilité pour assurer un basculement automatique sans interruption.

Vous pouvez activer les deux modèles de streaming sur la même table DynamoDB.

La vidéo suivante évoque plus en détail les différences entre les deux options.

Comparaison entre les flux DynamoDB et les flux de données Kinesis

Utilisation de Kinesis Data Streams pour récupérer les modifications apportées à DynamoDB

Vous pouvez utiliser Amazon Kinesis Data Streams pour capturer les modifications apportées à Amazon DynamoDB.

Kinesis Data Streams récupère les modifications au niveau élément dans n'importe quelle table DynamoDB et les réplique dans un flux de données Kinesis de votre choix. Vos applications peuvent accéder à ce flux et afficher les modifications au niveau élément en quasi-temps réel. Vous pouvez récupérer et stocker en continu des téraoctets de données par heure. Vous pouvez profiter d'un temps de conservation des données plus long et, grâce à une capacité de déploiement améliorée, vous pouvez atteindre simultanément deux applications en aval ou plus. Les autres avantages incluent des audits et une transparence de sécurité supplémentaires.

Kinesis Data Streams vous donne également accès à Amazon Data Firehose et au service géré Amazon pour Apache Flink. Ces services peuvent vous aider à créer des applications qui alimentent des tableaux de bord en temps réel, génèrent des alertes, implémentent une tarification et une publicité dynamiques, et mettent en œuvre une analytique des données sophistiquée et des algorithmes de machine learning.

> **Note**
>
> L'utilisation de Kinesis Data Streams pour DynamoDB est sujette à la tarification Kinesis Data Streams pour le flux de données, et à la tarification DynamoDB pour la table source.

Comment Kinesis Data Streams fonctionne avec DynamoDB

Lorsqu'un flux de données Kinesis est activé pour une table DynamoDB, la table envoie un enregistrement de données qui récupère toutes les modifications apportées aux données de cette table. Cet enregistrement de données comprend :

- L'heure spécifique à laquelle un élément a été récemment créé, mis à jour ou supprimé
- La clé primaire de cet élément
- Un instantané de l'enregistrement avant la modification
- Un instantané de l'enregistrement après la modification

Ces enregistrements de données sont capturés et publiés en quasi-temps réel. Une fois écrits dans le flux de données Kinesis, ils peuvent être lus comme tout autre enregistrement. Vous pouvez utiliser la bibliothèque cliente Kinesis, utiliser AWS Lambda, appeler l'API Kinesis Data Streams et utiliser d'autres services connectés. Pour en savoir plus, consultez Lecture de données à partir d'Amazon Kinesis Data Streams dans le Guide du développeur Amazon Kinesis Data Streams.

Ces modifications apportées aux données sont également récupérées de manière asynchrone. Kinesis n'a aucun impact sur les performances d'une table depuis laquelle elle est diffusée en continu. Les enregistrements de flux stockés dans votre flux de données Kinesis sont aussi chiffrés au repos. Pour en savoir plus, consultez Protection des données dans Amazon Kinesis Data Streams.

Les enregistrements de flux de données Kinesis peuvent s'afficher dans un ordre différent de celui dans lequel les modifications des éléments ont eu lieu. Les mêmes notifications d'éléments peuvent également apparaître plusieurs fois dans le flux. Vous pouvez vérifier l'attribut `ApproximateCreationDateTime` pour identifier l'ordre dans lequel les modifications des éléments ont eu lieu et les enregistrements en double.

Lorsque vous activez un flux de données Kinesis comme destination de streaming d'une table DynamoDB, vous pouvez configurer la précision des valeurs `ApproximateCreationDateTime`

en millisecondes ou microsecondes. Par défaut, `ApproximateCreationDateTime` indique l'heure de la modification en millisecondes. De plus, vous pouvez modifier cette valeur sur une destination de streaming active. Après cette mise à jour, les enregistrements de flux écrits dans Kinesis ont des valeurs `ApproximateCreationDateTime` de la précision souhaitée.

Les valeurs binaires écrites dans DynamoDB doivent être codées au format codé en base64. Toutefois, lorsque des enregistrements de données sont écrits dans un flux de données Kinesis, ces valeurs binaires codées sont codées une deuxième fois avec un codage base64. Lorsqu'il s'agit de lire ces enregistrements à partir d'un flux de données Kinesis, pour récupérer les valeurs binaires brutes, les applications doivent décoder ces valeurs deux fois.

DynamoDB facture l'utilisation de Kinesis Data Streams en unités de capture de données de modification. 1 Ko de modification par élément unique est considéré comme une unité de capture de données de modification. Le Ko de modification dans chaque élément est calculé par la plus grande des images « avant » et « après » de l'élément écrit dans le flux, en utilisant la même logique que la consommation d'unités de capacité pour les opérations d'écriture. Comme pour le fonctionnement du mode à la demande de DynamoDB, vous n'aurez pas besoin d'allouer le débit de capacité pour les unités de capture des données de modification.

Activer un flux de données Kinesis pour votre table DynamoDB

Vous pouvez activer ou désactiver le streaming vers Kinesis à partir de votre table DynamoDB existante à l'aide du AWS SDK ou du AWS Management Console().AWS Command Line InterfaceAWS CLI

- Vous ne pouvez diffuser des données de DynamoDB vers Kinesis Data Streams que dans le même AWS compte et dans AWS la même région que votre table.
- Vous pouvez uniquement diffuser des données depuis une table DynamoDB vers un flux de données Kinesis.

Modification d'une destination Kinesis Data Streams de votre table DynamoDB

Par défaut, tous les enregistrements de flux de données Kinesis incluent un attribut `ApproximateCreationDateTime`. Cet attribut représente un horodatage en millisecondes de l'heure approximative à laquelle chaque enregistrement a été créé. Vous pouvez modifier la précision de ces valeurs en utilisant https://console.aws.amazon.com/kinesis, le SDK ou le AWS CLI

Prise en main de Kinesis Data Streams pour Amazon DynamoDB

Cette section explique comment utiliser les tables Kinesis Data Streams pour Amazon DynamoDB avec la console Amazon DynamoDB, le () et l'API.AWS Command Line InterfaceAWS CLI

Création d’un flux de données Amazon Kinesis actif

Tous ces exemples utilisent la table DynamoDB `Music` créée dans le cadre du tutoriel Mise en route avec DynamoDB.

Pour en savoir plus sur la façon de créer des consommateurs et de connecter votre flux de données Kinesis à d’autres services AWS, consultez Lecture de données à partir de Kinesis Data Streams dans le Guide du développeur Amazon Kinesis Data Streams.

> **Note**
>
> Lorsque vous utilisez les partitions KDS pour la première fois, nous vous recommandons de les configurer de manière à ce que leur capacité s’adapte à la hausse ou à la baisse en fonction de vos modèles d’utilisation. Une fois que vous aurez accumulé davantage de données sur les modèles d’utilisation, vous pourrez ajuster les partitions de votre flux en conséquence.

Console

1. Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/kinesis/
2. Choisissez Create data stream (Créer un flux de données), puis suivez les instructions pour créer un flux appelé `samplestream`.
3. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
4. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
5. Choisissez la table Music.
6. Choisissez l’onglet Exportations et flux.
7. (Facultatif) Sous Informations sur le flux de données Amazon Kinesis, vous pouvez modifier la précision de l’horodatage des enregistrements de microseconde (par défaut) à milliseconde.
8. Choisissez samplestream dans la liste déroulante.

9. Cliquez sur le bouton Activer.

AWS CLI

1. Créez un flux de données Kinesis nommé `samplestream` à l'aide de la commande create-stream.

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

   Consultez Considérations relatives à la gestion des partitions pour Kinesis Data Streams avant de définir le nombre de partitions du flux de données Kinesis.

2. Vérifiez que le flux Kinesis est actif et prêt pour utilisation à l'aide de la commande describe-stream.

```
aws kinesis describe-stream --stream-name samplestream
```

3. Activez le streaming Kinesis sur la table DynamoDB à l'aide de la commande DynamoDB `enable-kinesis-streaming-destination`. Remplacez la valeur `stream-arn` par celle que la commande `describe-stream` a renvoyée à l'étape précédente. Activez éventuellement le streaming avec une précision plus détaillée (microseconde) des valeurs d'horodatage renvoyées sur chaque enregistrement.

   Activez le streaming avec une précision d'horodatage de l'ordre de la microseconde :

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
  --enable-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

   Ou activez le streaming avec la précision d'horodatage par défaut (milliseconde) :

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Vérifiez si le streaming Kinesis est actif sur la table à l'aide de la commande DynamoDB `describe-kinesis-streaming-destination`.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Écrivez des données dans la table DynamoDB à l'aide de la commande `put-item`, comme décrit dans le Guide du développeur DynamoDB.

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
 Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
```

6. Utilisez la commande CLI get-records de Kinesis pour extraire le contenu du flux Kinesis. Ensuite, utilisez l'extrait de code suivant pour désérialiser le contenu du flux.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
```

```
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. Suivez les instructions du Guide du développeur Kinesis Data Streams pour créer un flux de données Kinesis nommé `samplestream` à l'aide de Java.

   Consultez Considérations relatives à la gestion des partitions pour Kinesis Data Streams avant de définir le nombre de partitions pour le flux de données Kinesis.

2. Utilisez l'extrait de code suivant pour activer le streaming Kinesis sur la table DynamoDB. Activez éventuellement le streaming avec une précision plus détaillée (microseconde) des valeurs d'horodatage renvoyées sur chaque enregistrement.

   Activez le streaming avec une précision d'horodatage de l'ordre de la microseconde :

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();
```

```
EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

Ou activez le streaming avec la précision d'horodatage par défaut (milliseconde) :

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Suivez les instructions du Guide du développeur Kinesis Data Streams pour lire le flux de données créé.
4. Utilisez l'extrait de code suivant pour désérialiser le contenu du flux

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}
```

```
private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Modification d’un flux de données Amazon Kinesis actif

Cette section décrit comment apporter des modifications à une configuration Kinesis Data Streams pour DynamoDB active à l'aide de la console et de l'API.AWS CLI

AWS Management Console

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/
2. Accédez à votre table.
3. Choisissez l’onglet Exportations et flux.

AWS CLI

1. Appelez `describe-kinesis-streaming-destination` pour vérifier que le stream est `ACTIVE`.
2. Appelez `UpdateKinesisStreamingDestination`, comme dans cet exemple :

```
aws dynamodb update-kinesis-streaming-destination --table-name
 enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
enable_test_stream --update-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

3. Appelez `describe-kinesis-streaming-destination` pour vérifier que le stream est `UPDATING`.
4. Appelez `describe-kinesis-streaming-destination` régulièrement jusqu’à ce que le statut de streaming redevienne `ACTIVE`. Les mises à jour de précision de l’horodatage prennent

effet généralement sous 5 minutes. Une fois ce statut mis à jour, cela indique que la mise à jour est terminée et que la nouvelle valeur de précision sera appliquée aux futurs enregistrements.

5. Écrivez dans la table à l'aide de `putItem`.
6. Utilisez la commande `get-records` de Kinesis pour récupérer le contenu du flux.
7. Vérifiez que la valeur `ApproximateCreationDateTime` des écritures possède la précision souhaitée.

API Java

1. Fournissez un extrait de code qui construit une demande `UpdateKinesisStreamingDestination` et une réponse `UpdateKinesisStreamingDestination`.
2. Fournissez un extrait de code qui construit une demande `DescribeKinesisStreamingDestination` et une réponse `DescribeKinesisStreamingDestination response`.
3. Appelez `describe-kinesis-streaming-destination` régulièrement jusqu'à ce que le statut de streaming redevienne `ACTIVE`, indiquant que la mise à jour est terminée et que la nouvelle valeur de précision sera appliquée aux futurs enregistrements.
4. Effectuez des écritures sur la table.
5. Lisez le flux et désérialisez son contenu.
6. Vérifiez que la valeur `ApproximateCreationDateTime` des écritures possède la précision souhaitée.

Utilisation de partitions et de métriques avec DynamoDB Streams et Kinesis Data Streams

Considérations relatives à la gestion des partitions pour Kinesis Data Streams

Un flux de données Kinesis compte son débit dans les partitions. Dans Amazon Kinesis Data Streams, vous pouvez choisir entre le mode à la demande et le mode provisionné pour vos flux de données.

Nous vous recommandons d'utiliser le mode à la demande pour votre flux de données Kinesis si votre charge de travail d'écriture DynamoDB est très variable et imprévisible. En mode à la demande, aucune planification de la capacité n'est nécessaire, car Kinesis Data Streams gère automatiquement les partitions afin de fournir le débit nécessaire.

En cas de charges de travail prévisibles, vous pouvez utiliser le mode provisionné pour votre flux de données Kinesis. En mode provisionné, vous devez spécifier le nombre de partitions du flux de données nécessaires pour accueillir les enregistrements de capture des données de modification provenant de DynamoDB. Pour déterminer le nombre de partitions dont le flux de données Kinesis aura besoin pour prendre en charge votre table DynamoDB, vous devez disposer des valeurs d'entrée suivantes :

- Taille moyenne d'enregistrement de votre table DynamoDB en octets (`average_record_size_in_bytes`).
- Nombre maximum d'opérations d'écriture que votre table DynamoDB effectuera par seconde. Ce nombre inclut les opérations de création, de suppression et de mise à jour effectuées par vos applications, ainsi que les opérations générées automatiquement, comme les opérations de suppression générées par Time-to-Live (`write_throughput`).
- Pourcentage d'opérations de mise à jour et de remplacement que vous effectuez sur votre table par rapport aux opérations de création ou de suppression (`percentage_of_updates`). N'oubliez pas que les opérations de mise à jour et de remplacement répliquent les anciennes et les nouvelles images de l'élément modifié dans le flux. Elles génèrent ainsi deux fois la taille de l'élément DynamoDB.

Vous pouvez calculer le nombre de partitions (`number_of_shards`) dont votre flux de données Kinesis a besoin à l'aide des valeurs d'entrée de la formule suivante :

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Par exemple, vous pouvez avoir un débit maximal de 1 040 opérations d'écriture par seconde (`write_throughput`) avec une taille d'enregistrement moyenne de 800 octets (`average_record_size_in_bytes`). Si 25 % de ces opérations d'écriture sont des opérations de mise à jour (`percentage_of_updates`), vous aurez besoin de deux partitions (`number_of_shards`) pour accueillir votre débit de streaming DynamoDB :

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Tenez compte des points suivants avant d'utiliser la formule de calcul du nombre de partitions requises en mode provisionné pour les flux de données Kinesis :

- Cette formule permet d'estimer le nombre de partitions nécessaires pour accueillir vos enregistrements de données de modification DynamoDB. Elle ne représente pas le nombre total de partitions dont votre flux de données Kinesis a besoin, comme le nombre de partitions requises pour prendre en charge des consommateurs de flux de données Kinesis supplémentaires.
- Vous pouvez toujours rencontrer des exceptions de débit de lecture et d'écriture en mode provisionné si vous ne configurez pas votre flux de données pour gérer votre débit maximal. Dans ce cas, vous devez mettre à l'échelle manuellement votre flux de données pour l'adapter à votre trafic de données.
- Cette formule prend en compte le gonflement supplémentaire généré par DynamoDB avant de diffuser les enregistrements de données du journal des modifications à Kinesis Data Streams.

Pour en savoir plus sur les modes de capacité de Kinesis Data Streams, consultez Choix du mode de capacité du flux de données. Pour en savoir plus sur les différences tarifaires entre les différents modes de capacité, consultez la Tarification d'Amazon Kinesis Data Streams.

Surveiller la récupération de données de modification avec Kinesis Data Streams

DynamoDB fournit plusieurs indicateurs CloudWatch Amazon pour vous aider à surveiller la réplication de la capture des données de modification vers Kinesis. Pour une liste complète des CloudWatch indicateurs, voirMétriques et dimensions DynamoDB.

Afin de déterminer si votre flux dispose d'une capacité suffisante, nous vous recommandons de contrôler les éléments suivants, tant pendant l'activation du flux qu'en production :

- `ThrottledPutRecordCount` : nombre d'enregistrements limités par votre flux de données Kinesis en raison d'une capacité insuffisante de flux de données Kinesis. La valeur `ThrottledPutRecordCount` devrait rester aussi basse que possible, quoique vous pourriez rencontrer une certaine limitation lors de pics d'utilisation exceptionnels. DynamoDB réessaie d'envoyer des enregistrements limités dans le flux de données Kinesis, mais cela peut engendrer une latence de réplication plus élevée.

  Si vous rencontrez une limitation excessive et régulière, il se peut que vous deviez augmenter le nombre de partitions de flux Kinesis en proportion du débit d'écriture observé de votre table. Pour en savoir plus sur la détermination de la taille d'un flux de données Kinesis, consultez Détermination de la taille initiale d'un flux de données Kinesis.
- `AgeOfOldestUnreplicatedRecord` : temps écoulé depuis que la modification au niveau élément la plus ancienne restant à répliquer vers le flux de données Kinesis est apparue

dans la table DynamoDB. Dans des conditions normales de fonctionnement, la valeur `AgeOfOldestUnreplicatedRecord` devrait être de l'ordre de quelques millisecondes. Ce nombre augmente en fonction des tentatives de réplication infructueuses, lorsque celles-ci sont causées par des choix de configuration contrôlés par le client.

Si la métrique `AgeOfOldestUnreplicatedRecord` dépasse 168 heures, la réplication des modifications apportées au niveau des éléments de la table DynamoDB dans le flux de données Kinesis est automatiquement désactivée.

Les configurations contrôlées par le client qui peuvent entraîner des tentatives de réplication infructueuses sont, par exemple, une capacité de flux de données Kinesis sous-allouée causant une limitation excessive ou une mise à jour manuelle des stratégies d'accès de votre flux de données Kinesis empêchant DynamoDB d'ajouter des données à votre flux de données. Vous devrez peut-être allouer une capacité de flux de données Kinesis suffisante et vous assurer que les autorisations de DynamoDB sont inchangées pour que cette métrique reste aussi faible que possible.

- `FailedToReplicateRecordCount` : nombre d'enregistrements que DynamoDB n'a pas pu répliquer sur votre flux de données Kinesis. Certains éléments de plus de 34 Ko peuvent augmenter en taille pour modifier les enregistrements de données supérieurs à la limite de taille d'élément de 1 Mo de Kinesis Data Streams. Cette augmentation de taille se produit lorsque les éléments de plus de 34 Ko incluent un grand nombre de valeurs d'attribut booléennes ou vides. Les valeurs d'attribut booléennes et vides sont stockées sous la forme d'un octet dans DynamoDB. Toutefois, elles peuvent atteindre 5 octets lorsqu'elles sont sérialisées à l'aide de JSON standard pour la réplication Kinesis Data Streams. DynamoDB ne peut pas répliquer de tels enregistrements de modification dans votre flux de données Kinesis. DynamoDB ignore ces enregistrements de données de modification et continue automatiquement la réplication des enregistrements suivants.

Vous pouvez créer des CloudWatch alarmes Amazon qui envoient un message Amazon Simple Notification Service (Amazon SNS) pour notification lorsque l'une des mesures précédentes dépasse un seuil spécifique.

Utilisation de politiques IAM personnalisées pour Amazon Kinesis Data Streams et Amazon DynamoDB

La première fois que vous activez Amazon Kinesis Data Streams pour Amazon DynamoDB, DynamoDB crée automatiquement Gestion des identités et des accès AWS un rôle lié à un service (IAM) pour vous. Ce rôle, `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`,

permet à DynamoDB de gérer pour vous la réplication des modifications au niveau élément dans Kinesis Data Streams. Ne supprimez pas ce rôle lié à un service.

Pour en savoir plus sur l'utilisation des rôles liés à un service, consultez Utilisation des rôles liés à un service dans le Guide de l'utilisateur IAM.

> **Note**
>
> DynamoDB ne prend pas en charge les conditions basées sur une balise pour les politiques IAM.

Pour activer Amazon Kinesis Data Streams pour Amazon DynamoDB, vous devez disposer des autorisations suivantes sur la table.

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Pour décrire Amazon Kinesis Data Streams pour Amazon DynamoDB pour une table DynamoDB donnée, vous devez disposer des autorisations suivantes sur la table.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Pour désactiver Amazon Kinesis Data Streams pour Amazon DynamoDB, vous devez disposer des autorisations suivantes sur la table.

- `dynamodb:DisableKinesisStreamingDestination`

Pour mettre à jour Amazon Kinesis Data Streams pour Amazon DynamoDB, vous devez disposer des autorisations suivantes sur la table.

- `dynamodb:UpdateKinesisStreamingDestination`

Les exemples suivants montrent comment utiliser des politiques IAM afin d'accorder des autorisations pour Amazon Kinesis Data Streams pour Amazon DynamoDB.

Exemple : Activer Amazon Kinesis Data Streams pour Amazon DynamoDB

La politique IAM suivante octroie les autorisations nécessaires pour activer Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`. Elle n'accorde pas les autorisations permettant de désactiver, de mettre à jour ou de décrire Kinesis Data Streams pour DynamoDB pour la table `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Exemple : mise à jour Amazon Kinesis Data Streams pour Amazon DynamoDB

La politique IAM suivante octroie les autorisations nécessaires pour mettre à jour Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`. Elle n'accorde pas les autorisations permettant d'activer, de désactiver ou de décrire Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Exemple : Désactiver Amazon Kinesis Data Streams pour Amazon DynamoDB

La politique IAM suivante octroie les autorisations nécessaires pour désactiver Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`. Elle n'accorde pas les autorisations permettant d'activer, de mettre à jour ou de décrire Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
```

```
        }
    ]
}
```

Exemple : Appliquer de manière sélective des autorisations pour Amazon Kinesis Data Streams pour Amazon DynamoDB en fonction de la ressource

La politique IAM suivante octroie les autorisations nécessaires pour activer et décrire Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Music`, et n'accorde pas les autorisations permettant de désactiver Amazon Kinesis Data Streams pour Amazon DynamoDB pour la table `Orders`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

Utilisation de rôles liés à un service pour Kinesis Data Streams pour DynamoDB

Amazon Kinesis Data Streams pour Amazon Gestion des identités et des accès AWS DynamoDB utilise des rôles liés à un service (IAM). Un rôle lié à un service est un type unique de rôle IAM lié

directement à Kinesis Data Streams pour DynamoDB. Les rôles liés à un service sont prédéfinis par Kinesis Data Streams for DynamoDB et incluent toutes les autorisations requises par le service pour appeler d'autres services en votre nom.AWS

Un rôle lié à un service simplifie la configuration de Kinesis Data Streams pour DynamoDB, car vous n'avez pas besoin d'ajouter manuellement les autorisations nécessaires. Kinesis Data Streams pour DynamoDB définit les autorisations de ses rôles liés à un service et, sauf définition contraire, seul Kinesis Data Streams pour DynamoDB peut endosser ses rôles. Les autorisations définies comprennent la politique d'approbation et la politique d'autorisation. De plus, cette politique d'autorisation ne peut pas être attachée à une autre entité IAM.

Pour plus d'informations sur les autres services qui prennent en charge les rôles liés à un service, veuillez consulter Services AWS qui fonctionnent avec IAM et recherchez les services où Oui figure dans la colonne Rôle lié à un service. Choisissez un Oui ayant un lien permettant de consulter les détails du rôle pour ce service.

Autorisations de rôle lié à un service pour Kinesis Data Streams pour DynamoDB

Kinesis Data Streams for DynamoDB utilise le rôle lié à un service nommé. AWSServiceRoleForDynamoDBKinesisDataStreamsReplication L'objectif du rôle lié à un service est d'autoriser Amazon DynamoDB à gérer pour vous la réplication des modifications au niveau élément dans Kinesis Data Streams.

Le rôle lié à un service `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` approuve les services suivants pour endosser le rôle :

- `kinesisreplication.dynamodb.amazonaws.com`

La politique d'autorisations de rôle permet à Kinesis Data Streams pour DynamoDB d'accomplir les actions suivantes sur les ressources spécifiées :

- Action : `Put records and describe` sur `Kinesis stream`
- Action : `Generate data keys` activée pour placer les données `AWS KMS` dans les flux Kinesis chiffrés à l'aide de clés générées par l'utilisateur AWS KMS.

Pour le contenu exact du document de politique, voir Dynamo. DBKinesis ReplicationServiceRolePolicy

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Pour en savoir plus, consultez Service-Linked Role Permissions (autorisations du rôle lié à un service) dans le Guide de l'utilisateur IAM.

Création d'un rôle lié à un service pour Kinesis Data Streams pour DynamoDB

Vous n'avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous activez Kinesis Data Streams pour DynamoDB dans, dans ou dans AWS Management Console l'API,AWS CLIAWS Kinesis Data Streams for DynamoDB crée pour vous le rôle lié au service.

Si vous supprimez ce rôle lié à un service et que vous avez ensuite besoin de le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous activez Kinesis Data Streams pour DynamoDB, le service crée pour vous le rôle lié à un service.

Modification d'un rôle lié à un service pour Kinesis Data Streams pour DynamoDB

Kinesis Data Streams pour DynamoDB ne vous permet pas de modifier le rôle lié à un service `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Après avoir créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence à ce rôle. Néanmoins, vous pouvez modifier la description du rôle à l'aide d'IAM. Pour en savoir plus, consultez Modification d'un rôle lié à un service dans le Guide de l'utilisateur IAM.

Suppression d'un rôle lié à un service pour Kinesis Data Streams pour DynamoDB

Vous pouvez également utiliser la console IAM AWS CLI ou l'AWS API pour supprimer manuellement le rôle lié à un service. Pour ce faire, vous devez commencer par nettoyer les ressources de votre rôle lié à un service. Vous pouvez ensuite supprimer ce rôle manuellement.

> **Note**
>
> Si le service Kinesis Data Streams pour DynamoDB utilise le rôle lorsque vous essayez de supprimer les ressources, la suppression peut échouer. Si cela se produit, patientez quelques minutes et réessayez.

Pour supprimer manuellement le rôle lié au service à l'aide d'IAM

Utilisez la console IAM, le AWS CLI, ou l'AWS API pour supprimer le rôle lié au `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` service. Pour en savoir plus, consultez Suppression d'un rôle lié à un service dans le Guide de l'utilisateur IAM.

Modifier la récupération de données pour DynamoDB Streams

DynamoDB Streams récupère une séquence chronologique des modifications au niveau élément dans toute table DynamoDB et stocke ces informations dans un journal pendant jusqu'à 24 heures. Les applications ont accès à ce journal et affichent les éléments de données à mesure qu'ils s'affichent avant et après qu'ils ont été modifiés, pratiquement en temps réel.

Le chiffrement au repos chiffre les données dans les flux DynamoDB Streams. Pour en savoir plus, consultez Chiffrement de DynamoDB au repos.

Un flux DynamoDB est un flux ordonné d'informations sur les modifications apportées aux éléments d'une table DynamoDB. Lorsque vous activez un flux sur une table, DynamoDB récupère des informations sur chaque modification apportée à des éléments de données dans la table.

Chaque fois qu'une application crée, met à jour ou supprime des éléments dans la table, DynamoDB Streams écrit un enregistrement de flux avec le ou les attributs de clé primaire des éléments qui ont été modifiés. Un registre de flux contient des informations sur une modification de données dans un seul élément d'une table DynamoDB. Vous pouvez configurer le flux de telle sorte que les enregistrements de flux récupèrent des informations supplémentaires, telles que les images « avant » et « après » d'éléments modifiés.

DynamoDB Streams permet de s'assurer de ce qui suit :

- Chaque enregistrement de flux s'affiche exactement une fois dans le flux.
- Pour chaque élément modifié dans une table DynamoDB, les enregistrements de flux apparaissent dans l'ordre des modifications réelles.

DynamoDB Streams écrit les enregistrements de flux en quasi-temps réel, afin que vous puissiez créer des applications qui consomment ces flux et entreprennent des actions basées sur le contenu.

Rubriques

- Points de terminaison pour DynamoDB Streams
- Activation d'un flux
- Lecture et traitement de flux
- DynamoDB Streams et time-to-live
- Utilisation de l'adaptateur DynamoDB Streams Kinesis pour traiter des enregistrements de flux

- API de bas niveau DynamoDB Streams : exemple Java
- Streams et déclencheurs DynamoDB AWS Lambda
- DynamoDB Streams et Apache Flink

Points de terminaison pour DynamoDB Streams

AWS gère des points de terminaison distincts pour DynamoDB et DynamoDB Streams. Pour utiliser des tables et index de base de données, votre application doit accéder à un point de terminaison DynamoDB. Pour lire et traiter des enregistrements DynamoDB Streams, votre application doit pouvoir accéder à un point de terminaison DynamoDB Streams dans la même région.

DynamoDB Streams propose deux ensembles de points de terminaison. Il s'agit des options suivantes :

- IPv4-points de terminaison uniquement : points de terminaison dotés de la `streams.dynamodb.<region>.amazonaws.com` convention de dénomination.
- Points de terminaison à double pile : nouveaux points de terminaison compatibles avec les deux IPv4 IPv6 et conformes à la `streams-dynamodb.<region>.api.aws` convention de dénomination.

> **Note**
>
> Pour accéder à la liste complète des régions et points de terminaison DynamoDB et DynamoDB Streams, consultez Régions et points de terminaison dans Références générales AWS.

Ils AWS SDKs fournissent des clients distincts pour DynamoDB et DynamoDB Streams. En fonction de vos exigences, votre application peut accéder à un point de terminaison DynamoDB, un point de terminaison DynamoDB Streams, ou aux deux en même temps. Pour vous connecter aux deux points de terminaison, votre application doit instancier deux clients, l'un pour DynamoDB, et l'autre pour DynamoDB Streams.

Activation d'un flux

Vous pouvez activer un flux sur une nouvelle table lorsque vous le créez à l'aide du AWS CLI ou de l'un des AWS SDKs. Vous pouvez également activer ou désactiver un flux sur une table existante, ou

modifier les paramètres d'un flux. DynamoDB Streams opérant de manière asynchrone, l'activation d'un flux n'a aucune incidence sur les performances d'une table.

La manière la plus simple de gérer DynamoDB Streams consiste à utiliser l'AWS Management Console.

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le tableau de bord de la console DynamoDB, choisissez Tables, puis sélectionnez une table.
3. Choisissez l'onglet Exportations et flux.
4. Dans la section Détails du flux DynamoDB, choisissez Activer.
5. Sur la page Activer le flux DynamoDB, choisissez les informations qui seront écrites dans le flux chaque fois que des données de la table seront modifiées :

   - Attributs de clés uniquement – Uniquement les attributs de clé de l'élément modifié.
   - New image (Nouvelle image) – L'élément entier, tel qu'il apparaît après sa modification.
   - Old image (Ancienne image) – L'élément entier, tel qu'il apparaissait avant sa modification.
   - New and old images (Nouvelle et ancienne images) – La nouvelle image et l'ancienne image de l'élément.

   Lorsque les paramètres vous conviennent, choisissez Activer le streaming.
6. (Facultatif) Pour désactiver un flux existant, choisissez Désactiver sous Détails du flux DynamoDB.

Vous pouvez également utiliser les API `CreateTable` ou `UpdateTable` pour activer ou modifier un flux. Le paramètre `StreamSpecification` détermine la façon dont le flux est configuré :

- `StreamEnabled` – Spécifique si un flux de données est activé (`true`) ou désactivé (`false`) pour la table.
- `StreamViewType` – Spécifie les informations à écrire dans le flux à chaque modification des données de la table :
  - `KEYS_ONLY` – Uniquement les attributs de clé de l'élément modifié.
  - `NEW_IMAGE` – L'élément entier, tel qu'il apparaît après sa modification.
  - `OLD_IMAGE` – L'élément entier, tel qu'il apparaissait avant sa modification.

- `NEW_AND_OLD_IMAGES` – La nouvelle image et l'ancienne image de l'élément.

Vous pouvez activer ou désactiver un flux à tout moment. Cependant, vous recevez un `ValidationException` si vous essayez d'activer un flux sur une table qui en possède déjà un. Vous recevez également un `ValidationException` si vous essayez de désactiver un flux sur une table qui n'en possède pas.

Lorsque vous définissez `StreamEnabled` sur `true`, DynamoDB crée un flux avec un descripteur de flux unique qui lui est attribué. Si vous désactivez et puis réactivez un flux sur la table, un flux est créé avec un descripteur de flux différent.

Chaque flux est identifié de manière unique par un Amazon Resource Name (ARN). Voici un exemple d'ARN pour un flux sur une table DynamoDB nommée `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Pour déterminer le dernier descripteur de flux pour une table, émettez une demande DynamoDB `DescribeTable`, puis recherchez l'élément `LatestStreamArn` dans la réponse.

> **Note**
>
> Il n'est pas possible de modifier un `StreamViewType` une fois qu'un flux a été configuré. Si vous devez apporter des modifications à un flux après sa configuration, vous devez désactiver le flux actuel et en créer un nouveau.

Lecture et traitement de flux

Pour lire et traiter un flux, votre application doit se connecter à un point de terminaison DynamoDB Streams et émettre des demandes d'API.

Un flux se compose d'enregistrements de flux. Chaque enregistrement de flux représente une modification de donnée unique dans la table DynamoDB à laquelle le flux appartient. Chaque enregistrement de flux se voit attribuer un numéro de séquence, ce qui reflète l'ordre dans lequel l'enregistrement a été publié dans le flux.

Les enregistrements de flux sont organisés en groupes, ou partitions. Chaque partition agit comme un conteneur pour plusieurs enregistrements de flux et contient les informations requises pour

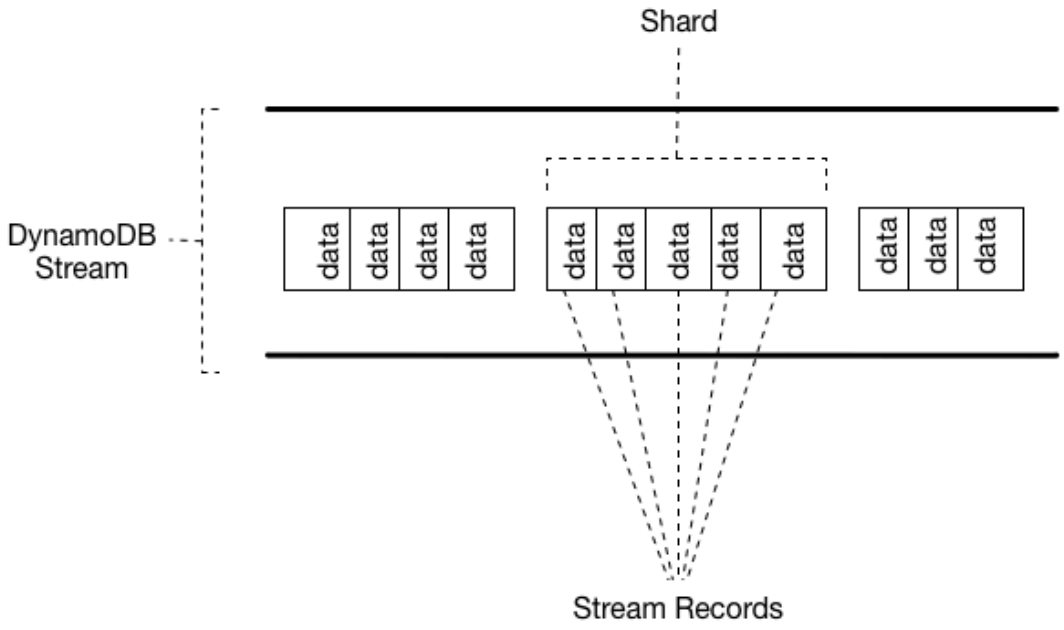
accéder à ces enregistrements et les itérer. Les enregistrements de flux au sein d'une partition sont automatiquement supprimés au bout de 24 heures.

Les partitions sont éphémères. Elles sont créées et supprimées automatiquement, en fonction des besoins. Toute partition peut également se diviser en plusieurs partitions nouvelles. Cela se produit également automatiquement. (Notez qu'il est aussi possible pour une partition parent d'avoir une seule partition enfant). Une partition peut se diviser en réponse à des niveaux élevés d'activité d'écriture sur sa table parent, de telle sorte que les applications puissent traiter les enregistrements issus de plusieurs partitions en parallèle.

Si vous désactivez un flux, toute partition ouverte sera fermée. Les données du flux restent lisibles pendant 24 heures.

Comme les partitions ont une lignée (parent et enfants), une application doit toujours traiter une partition parent avant de traiter une partition enfant. Cela garantit que les enregistrements de flux sont également traités dans l'ordre adéquat. (Si vous utilisez l'adaptateur DynamoDB Streams Kinesis, cela est géré automatiquement. Votre application traite les partitions et les enregistrements de flux dans l'ordre correct. Elle gère automatiquement les partitions nouvelles ou ayant expiré, en plus des partitions qui ont été scindées pendant l'exécution de l'application. Pour plus d'informations, consultez Utilisation de l'adaptateur DynamoDB Streams Kinesis pour traiter des enregistrements de flux.)

Le schéma suivant illustre la relation entre un flux de données, les partitions dans le flux et les enregistrements de flux dans les partitions.



> **Note**
>
> Si vous effectuez une opération `PutItem` ou `UpdateItem` qui ne modifie aucune donnée dans un élément, DynamoDB Streams n'écrit pas d'enregistrement de flux pour cette opération.

Pour accéder à un flux de données et traiter les enregistrements de flux qu'il contient, vous devez effectuer les opérations suivantes :

- Déterminer l'Amazon Resource Name (ARN) unique du flux auquel vous souhaitez accéder.
- Déterminer quelles sont la ou les partitions du flux qui contiennent les enregistrements de flux qui vous intéressent.
- Accéder aux partitions et récupérer les enregistrements de flux que vous voulez.

> **Note**
>
> Deux processus au maximum doivent lire simultanément à partir de la même partition de flux. Avoir plus de 2 lecteurs par partition peut entraîner une limitation.

L'API DynamoDB Streams fournit les actions suivantes à l'usage des programmes d'application :

- `ListStreams` – Renvoie la liste des descripteurs de flux pour le compte et le point de terminaison actuels. Vous pouvez en option demander uniquement les descripteurs de flux pour un nom de table particulier.
- `DescribeStream` – Renvoie des informations sur un flux, y compris le statut actuel du flux, son Amazon Resource Name (ARN), la composition de ses partitions et sa table DynamoDB correspondante. Vous pouvez éventuellement récupérer la partition enfant associée à la partition parent à l'aide du champ `ShardFilter`.
- `GetShardIterator` – Renvoie un itérateur de partition décrivant un emplacement au sein d'une partition. Vous pouvez demander que l'itérateur fournisse un accès au point le plus ancien, au point le plus récent ou à un point particulier dans le flux.
- `GetRecords` – Renvoie les enregistrements de flux à partir d'une partition donnée. Vous devez fournir l'itérateur de partition renvoyé à partir d'une requête `GetShardIterator`.

Pour obtenir une description complète de ces opérations d'API, y compris des exemples de demandes et de réponses, consultez la Référence des API Amazon DynamoDB Streams.

Découverte de partitions

Découvrez de nouvelles partitions dans votre flux DynamoDB à l'aide de deux méthodes puissantes. En tant qu'utilisateur d'Amazon DynamoDB Streams, vous disposez de deux méthodes efficaces pour suivre et identifier de nouvelles partitions :

Interrogation de l'ensemble de la topologie du flux

Interrogez régulièrement le flux à l'aide de l'API `DescribeStream`. Celle-ci renvoie toutes les partitions du flux, y compris celles qui ont été créées. En comparant les résultats au fil du temps, vous pouvez détecter les partitions ajoutées récemment.

Découverte de partitions enfants

Recherchez un sous-ensemble de partitions à l'aide de l'API `DescribeStream` avec le paramètre `ShardFilter`. En spécifiant une partition parent dans la demande, DynamoDB Streams renvoie ses partitions enfants immédiates. Cette approche est utile lorsque vous avez uniquement besoin de suivre la lignée des partitions sans analyser l'intégralité du flux.

Les applications consommant des données de DynamoDB Streams peuvent passer efficacement de la lecture d'une partition fermée à sa partition enfant à l'aide du paramètre `ShardFilter`, évitant ainsi des appels répétés à l'API `DescribeStream` afin de récupérer et de parcourir la carte des partitions à la recherche des partitions fermées et ouvertes. Vous pouvez ainsi découvrir rapidement les partitions enfants après la fermeture d'une partition parent, ce qui rend vos applications de traitement de flux plus réactives et plus économiques.

Les deux méthodes vous permettent de suivre l'évolution de la structure de vos flux DynamoDB, afin de ne jamais manquer les mises à jour de données ou les modifications critiques de partitions.

Limite de conservation des données pour DynamoDB Streams

Toutes les données dans DynamoDB Streams ont un time-to-live de 24 heures. Vous pouvez extraire et analyser les 24 dernières heures d'activité d'une table donnée. Cependant, les données datant de plus de 24 heures sont susceptibles d'être supprimées à tout moment.

Si vous désactivez un flux sur une table, les données du flux continueront d'être lisibles pendant 24 heures. Passé ce délai, les données expirent et les enregistrements de flux sont supprimés

automatiquement. Il n'existe pas de mécanisme pour supprimer manuellement un flux existant. Vous devez attendre que la limite de rétention expire (24 heures) et tous les enregistrements de flux seront supprimés.

DynamoDB Streams et time-to-live

Vous pouvez sauvegarder, ou traiter de toute autre façon, les éléments supprimés par Time-to-live (TTL) en activant Amazon DynamoDB Streams sur la table et en traitant les enregistrements de flux des éléments expirés. Pour en savoir plus, consultez Lecture et traitement de flux.

L'enregistrement de flux contient un champ d'identité utilisateur `Records[<index>].userIdentity`.

Les éléments supprimés par le processus Time-to-live après expiration ont les champs suivants :

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> Lorsque vous utilisez le processus TTL dans une table globale, le champ `userIdentity` est défini dans la région dans laquelle ce processus a été effectué. Ce champ n'est pas défini dans les autres régions lorsque la suppression est répliquée.

Le JSON suivant montre la partie pertinente d'un seul enregistrement de flux.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }
```

```
        ...

    }
]
```

Utilisation de DynamoDB Streams et Lambda pour archiver les éléments supprimés TTL

La combinaison de la DynamoDB time-to-live (TTL), de DynamoDB Streams, et de AWS Lambda peut permettre de simplifier l'archivage des données, de réduire les coûts de stockage de DynamoDB et de réduire la complexité du code. L'utilisation de Lambda comme consommateur de flux offre de nombreux avantages, notamment la réduction des coûts par rapport à d'autres consommateurs tels que Kinesis Client Library (KCL). Vous n'êtes pas facturé pour les appels d'API `GetRecords` sur votre flux DynamoDB lorsque vous utilisez Lambda pour consommer des événements, et Lambda peut fournir un filtrage d'événements en identifiant les modèles JSON dans un événement de flux. Avec le filtrage de contenu des modèles d'événements, vous pouvez définir jusqu'à cinq filtres différents pour contrôler quels événements sont envoyés à Lambda en vue d'être traités. Cela permet de réduire les appels de vos fonctions Lambda, de simplifier le code et de réduire le coût global.

Alors que DynamoDB Streams contient toutes les modifications de données, telles que les actions `Create`, `Modify` et `Remove`, cela peut entraîner des appels indésirables de votre fonction Lambda d'archivage. Par exemple, supposons que vous ayez une table contenant 2 millions de modifications de données par heure dans le flux, mais que moins de 5 % d'entre elles soient des suppressions d'éléments qui expireront via le processus TTL et devront être archivées. Avec les filtres de source d'événement Lambda, la fonction Lambda n'effectue que 100 000 appels par heure. Il en résulte que le filtrage des événements vous est facturé uniquement pour les appels nécessaires au lieu des 2 millions d'appels que vous auriez sans le filtrage des événements.

Le filtrage des événements est appliqué au mappage de source d'événement Lambda, une ressource qui lit à partir d'un événement choisi (le flux DynamoDB) et appelle une fonction Lambda. Dans le diagramme suivant, vous pouvez voir comment un élément time-to-live supprimé est consommé par une fonction Lambda utilisant des flux et des filtres d'événements.

Modèle de filtrage d'événement DynamoDB time-to-live

L'ajout du JSON suivant à vos critères de filtrage du mappage de source d'événement permet l'appel de votre fonction Lambda uniquement pour les éléments supprimés TTL :

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

Création d'un mappage des sources d'AWS Lambdaévénements

Utilisez les extraits de code suivants pour créer un mappage de source d'événement filtré que vous pouvez connecter au flux DynamoDB d'une table. Chaque bloc de code inclut le modèle de filtre d'événement.

AWS CLI

```
aws lambda create-event-source-mapping \
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();
```

```
CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();

try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
}
```

Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
```

```
    console.error(err);
}
```

Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

Utilisation de l’adaptateur DynamoDB Streams Kinesis pour traiter des enregistrements de flux

L’utilisation de l’adaptateur Amazon Kinesis est la méthode recommandée pour consommer des flux d’Amazon DynamoDB. L’API DynamoDB Streams est volontairement semblable à celle de Kinesis Data Streams. Dans les deux services, les flux de données sont composés de partitions qui sont des conteneurs pour enregistrements de flux. Les deux services APIs contiennent`ListStreams`, `DescribeStream``GetShards`, et `GetShardIterator` opérations. (Si ces actions de DynamoDB Streams sont similaires à leurs homologues dans Kinesis Data Streams, elles ne sont pas identiques à 100 %.)

En tant qu’utilisateur de DynamoDB Streams, vous pouvez utiliser les modèles de conception figurant dans la KCL pour traiter les partitions et les enregistrements de flux de DynamoDB Streams. Pour ce faire, vous utilisez l’adaptateur Kinesis DynamoDB Streams. L’adaptateur Kinesis implémente l’interface Kinesis Data Streams afin que la KCL puisse être utilisée pour la consommation et le traitement des enregistrements de DynamoDB Streams. Pour obtenir des instructions sur la configuration et l'installation de l'adaptateur DynamoDB Streams Kinesis, consultez le référentiel. GitHub

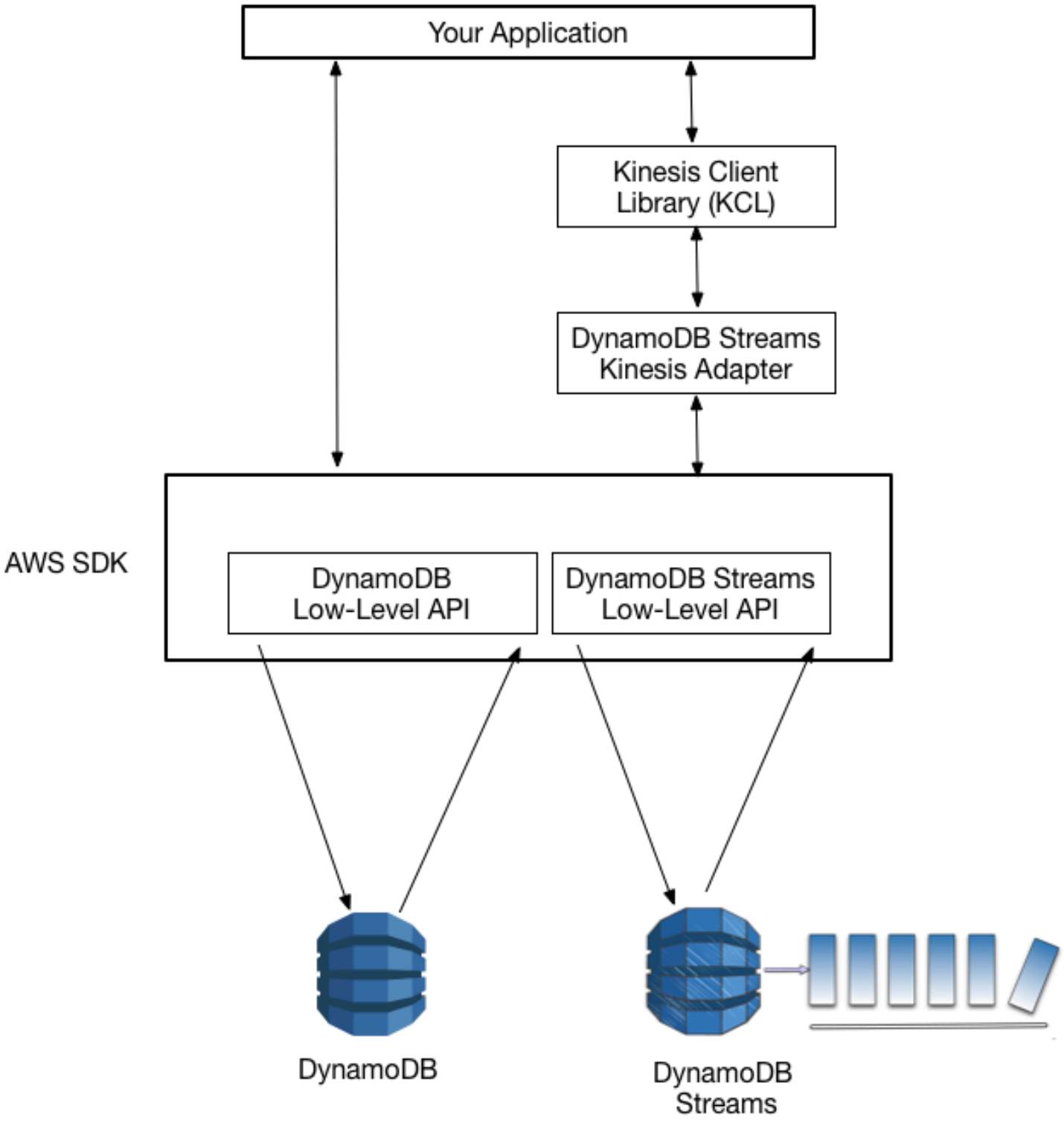
Vous pouvez écrire des applications pour Kinesis Data Streams à l’aide de la bibliothèque client Kinesis (KCL). La KCL simplifie le codage en fournissant des abstractions utiles par-dessus l’API Kinesis Data Streams de bas niveau. Pour en savoir plus sur la KCL, consultez Développement d’applications consommateur à l’aide de la bibliothèque client Kinesis dans le Guide du développeur Amazon Kinesis Data Streams.

DynamoDB recommande d'utiliser la version 3.x de KCL avec le SDK AWS pour Java v2.x. La version 1.x de l'adaptateur DynamoDB Streams Kinesis actuelle AWS avec SDK AWS SDK pour Java pour v1.x continuera d'être entièrement prise en charge tout au long de son cycle de vie, comme prévu pendant la période de transition, conformément à la politique de maintenance des outils.AWS SDKs

> **Note**
>
> Les versions 1.x et 2.x de la bibliothèque client Amazon Kinesis (KCL) sont obsolètes. KCL 1.x arrivera end-of-support le 30 janvier 2026. Nous vous recommandons vivement de migrer vos applications KCL utilisant la version 1.x vers la version la plus récente de la KCL avant le 30 janvier 2026. Pour trouver la dernière version de KCL, consultez la page de la bibliothèque client Amazon Kinesis sur. GitHub Pour en savoir plus sur la bibliothèque client Kinesis,

consultez Use Kinesis Client Library. Pour en savoir plus sur la migration de la KCL 1.x vers la KCL 3.x, consultez Migration de la KCL 1.x vers la KCL 3.x.

Le diagramme suivant illustre la manière dont ces bibliothèques interagissent entre elles.



Avec l'adaptateur Kinesis DynamoDB Streams en place, vous pouvez commencer à développer sur l'interface KCL, avec les appels d'API dirigés de manière transparente vers le point de terminaison DynamoDB Streams.

Lorsque votre application démarre, elle appelle la KCL pour instancier un worker. Vous devez fournir au travailleur les informations de configuration de l'application, telles que le descripteur de flux et les AWS informations d'identification, ainsi que le nom d'une classe de processeur d'enregistrements que vous fournissez. À mesure qu'il exécute le code dans le processeur d'enregistrements, le worker effectue les tâches suivantes :

- Se connecte au flux
- Énumère les partitions dans le flux
- Vérifie et énumère les partitions enfants d'une partition parent fermée dans le flux
- Coordonne les associations de partition avec les autres travaux (le cas échéant)
- Instancie un processeur d'enregistrements pour chaque partition qu'il gère
- Extrait des enregistrements du flux
- Pousse les enregistrements sur le processeur d'enregistrements correspondant
- Contrôle les enregistrements traités
- Équilibre les associations partition-worker lorsque le nombre d'instances de worker change
- Équilibre les associations partition-worker quand des partitions sont fractionnées

> **Note**
>
> Pour une description des concepts de KCL évoqués ici, consultez Développement d'applications consommateur à l'aide de la bibliothèque client Kinesis dans le Guide du développeur Amazon Kinesis Data Streams.
> Pour plus d'informations sur l'utilisation des flux avec,AWS Lambda voir Streams et déclencheurs DynamoDB AWS Lambda

Migration de la KCL 1.x vers la KCL 3.x

Présentation de

Ce guide fournit des instructions pour migrer votre application consommateur de la KCL 1.x vers la KCL 3.x. En raison des différences architecturales entre la KCL 1.x et la KCL 3.x, la migration nécessite la mise à jour de plusieurs composants pour garantir la compatibilité.

La KCL 1.x utilise des classes et interfaces différentes rapport à la KCL 3.x. Vous devez d'abord migrer le processeur d'enregistrements, la fabrique de processeurs d'enregistrements et les classes

de workers vers le format compatible avec la KCL 3.x, puis suivre les étapes de migration de la KCL 1.x vers la KCL 3.x.

Étapes de la migration

Rubriques

- Étape 1 : migrer le processeur d'enregistrements
- Étape 2 : migrer la fabrique de processeurs d'enregistrements
- Étape 3 : migrer le worker
- Étape 4 : présentation de la configuration de la KCL 3.x et recommandations
- Étape 5 : migrer de la KCL 2.x vers la KCL 3.x

Étape 1 : migrer le processeur d'enregistrements

L'exemple suivant illustre un processeur d'enregistrements implémenté pour l'adaptateur DynamoDB Streams Kinesis version 1.x :

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class StreamsRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }
```

```
    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                // record processing and checkpointing logic
            }
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

Pour migrer la RecordProcessor classe

1. Remplacez les interfaces `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` et `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat`

par `software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordP` comme suit :

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;

import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

2. Mettez à jour les instructions d'importation des méthodes `initialize` et `processRecords` :

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

// import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
```

3. Remplacez la méthode `shutdownRequested` par les nouvelles méthodes suivantes : `leaseLost`, `shardEnded` et `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...) and
 shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
```

```
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

Voici la version mise à jour de la classe du processeur d'enregistrements :

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
```

```
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
```

```
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> L'adaptateur DynamoDB Streams Kinesis utilise désormais le modèle Record. SDKv2 Dans SDKv2, `AttributeValue` les objets complexes (`BS`,`NS`,, `ML`,`SS`) ne renvoient jamais nul. Vérifiez si ces valeurs existent à l'aide des méthodes `hasBs()`, `hasNs()`, `hasM()`, `hasL()` et `hasSs()`.

Étape 2 : migrer la fabrique de processeurs d'enregistrements

La fabrique de processeurs d'enregistrements est responsable de la création des processeurs d'enregistrements lorsqu'un bail est acquis. Voici un exemple de fabrique KCL 1.x :

```
package com.amazonaws.codesamples;

import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

Pour migrer la **`RecordProcessorFactory`**

- Remplacez l'interface implémentée `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` par `software.amazon.kinesis.processor.ShardRecordProcessorFactory` comme suit :

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

Change the return signature for createProcessor.

// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Voici un exemple de fabrique de processeurs d'enregistrements dans 3.0 :

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor();
    }
}
```

Étape 3 : migrer le worker

Dans la version 3.0 de la KCL, une nouvelle classe, appelée Scheduler, remplace la classe Worker. Voici un exemple de worker KCL 1.x :

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

Pour migrer le worker

1. Modifiez l'instruction `import` de la classe `Worker` pour les instructions d'importation pour les classes `Scheduler` et `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Importez `StreamTracker` et remplacez l'importation `StreamsWorkerFactory` par `StreamsSchedulerFactory`.

   ```
   import software.amazon.kinesis.processor.StreamTracker;
   // import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
   import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
   ```

3. Choisissez la position à partir de laquelle vous souhaitez démarrer l'application. Vous avez le choix entre `TRIM_HORIZON` et `LATEST`.

   ```
   import software.amazon.kinesis.common.InitialPositionInStream;
   import software.amazon.kinesis.common.InitialPositionInStreamExtended;
   ```

4. Créez une instance `StreamTracker`.

   ```
   StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
           streamArn,
   ```

```
 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. Créez l'objet `AmazonDynamoDBStreamsAdapterClient`.

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. Créez l'objet `ConfigsBuilder`.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Créez le `Scheduler` à l'aide de `ConfigsBuilder`, comme illustré dans l'exemple suivant :

```


import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
```

```
import software.amazon.kinesis.coordinator.Scheduler;

...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();


DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);

// Use ConfigsBuilder to configure settings
RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

Important

Le paramètre `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` assure la compatibilité entre l'adaptateur DynamoDB Streams Kinesis pour la KCL v3 et la KCL v1, et non entre la KCL v2 et la KCL v3.

Étape 4 : présentation de la configuration de la KCL 3.x et recommandations

Pour obtenir une description détaillée des configurations introduites après la KCL 1.x qui sont pertinentes dans la KCL 3.x, consultez KCL configurations et KCL migration client configuration.

> **Important**
>
> Au lieu de créer directement des objets de `checkpointConfig`, `coordinatorConfig`, `leaseManagementConfig`, `metricsConfig`, `processorConfig` et `retrievalConfig`, nous vous recommandons de définir des configurations dans la KCL 3.x et versions ultérieures à l'aide de `ConfigsBuilder`, afin d'éviter les problèmes d'initialisation du Scheduler. `ConfigsBuilder` fournit une méthode plus flexible et plus facile à gérer pour configurer votre application KCL.

Configurations avec mise à jour de la valeur par défaut dans la KCL 3.x

`billingMode`

Dans la KCL version 1.x, la valeur par défaut de `billingMode` est définie sur `PROVISIONED`. En revanche, avec la KCL version 3.x, le `billingMode` par défaut est `PAY_PER_REQUEST` (mode à la demande). Nous vous recommandons d'utiliser le mode de capacité à la demande pour votre table de baux, afin d'ajuster automatiquement la capacité en fonction de votre utilisation. Pour obtenir des conseils sur l'utilisation de la capacité allouée pour vos tables de baux, consultez Best practices for the lease table with provisioned capacity mode.

`idleTimeBetweenReadsInMillis`

Dans la KCL version 1.x, la valeur par défaut de `idleTimeBetweenReadsInMillis` est définie sur 1 000 (soit 1 seconde). La KCL version 3.x définit la valeur par défaut de `idleTimeBetweenReadsInMillis` sur 1 500 (soit 1,5 seconde), mais l'adaptateur Amazon DynamoDB Streams Kinesis remplace la valeur par défaut par 1 000 (soit 1 seconde).

Nouvelles configurations de la KCL 3.x

`leaseAssignmentIntervalMillis`

Cette configuration définit l'intervalle de temps avant que les partitions récemment découvertes ne commencent à être traitées. Elle est calculée comme suit :

1,5 × `leaseAssignmentIntervalMillis`. Si ce paramètre n'est pas explicitement configuré, l'intervalle de temps est défini par défaut sur 1,5 × `failoverTimeMillis`. Le traitement des nouvelles partitions consiste à analyser la table de baux et à interroger un index secondaire global (GSI) de la table de baux. La réduction de `leaseAssignmentIntervalMillis` augmente la fréquence de ces opérations d'analyse et d'interrogation, ce qui entraîne une augmentation des coûts de DynamoDB. Nous vous recommandons de définir cette valeur sur 2 000 (soit 2 secondes) afin de réduire le délai de traitement des nouvelles partitions.

`shardConsumerDispatchPollIntervalMillis`

Cette configuration définit l'intervalle entre les interrogations successives effectuées par le consommateur de partitions pour déclencher des transitions d'état. Dans la KCL version 1.x, ce comportement était contrôlé par le paramètre `idleTimeInMillis`, qui n'était pas exposé en tant que paramètre configurable. Avec la KCL version 3.x, nous vous recommandons de définir cette configuration de sorte qu'elle corresponde à la valeur utilisée pour `idleTimeInMillis` dans votre configuration de la KCL version 1.x.

Étape 5 : migrer de la KCL 2.x vers la KCL 3.x

Pour garantir une transition fluide et une compatibilité avec la version la plus récente de la bibliothèque client Kinesis (KCL), suivez les étapes 5 à 8 des instructions du guide de migration pour la mise à niveau de la KCL 2.x vers la KCL 3.x.

Pour la résolution des problèmes courants liés à la KCL 3.x, consultez Troubleshooting KCL consumer applications.

Restauration par régression de la version précédente de la KCL

Cette rubrique explique comment restaurer la version précédente de la KCL pour votre application consommateur. Le processus de restauration par régression comprend deux étapes :

1. Exécuter l'outil de migration de la KCL
2. Redéployer le code de la version précédente de la KCL

Étape 1 : exécuter l'outil de migration de la KCL

Si vous avez besoin de restaurer la version précédente de la KCL, vous devez exécuter l'outil de migration de la KCL. L'outil effectue deux tâches importantes :

- Il supprime une table de métadonnées appelée table des métriques de worker, ainsi qu'un index secondaire global de la table des baux dans DynamoDB. Ces artefacts sont créés par la KCL 3.x, mais ils ne sont pas nécessaires lorsque vous restaurez la version précédente.
- Ainsi, tous les workers peuvent s'exécuter dans un mode compatible avec la KCL 1.x et commencer à utiliser l'algorithme d'équilibrage de charge utilisé dans les versions précédentes de la KCL. Si vous rencontrez des problèmes avec le nouvel algorithme d'équilibrage de charge dans la KCL 3.x, cela permettra de les résoudre immédiatement.

> **Important**
>
> La table des états de coordinateur de DynamoDB doit exister et ne doit pas être supprimée pendant les processus de migration, de restauration par régression et de restauration par progression.

> **Note**
>
> Il est important que tous les workers de votre application consommateur utilisent le même algorithme d'équilibrage de charge à un moment donné. L'outil de migration de la KCL s'assure que tous les workers de votre application consommateur KCL 3.x basculent vers le mode compatible avec la KCL 1.x, afin qu'ils exécutent le même algorithme d'équilibrage de charge lors de la restauration par régression de l'application vers la version précédente de la KCL.

Vous pouvez télécharger l'outil de migration KCL dans le répertoire des scripts du référentiel KCL GitHub. Exécutez le script à partir d'un worker ou d'un hôte disposant des autorisations appropriées pour écrire dans la table des états de coordinateur, la table des métriques de worker et la table des baux. Assurez-vous que les autorisations IAM appropriées sont configurées pour les applications consommateur KCL. Exécutez le script une seule fois par application KCL à l'aide de la commande spécifiée :

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parameters

`--region`

*region*Remplacez-le par votre Région AWS.

`--application_name`

Ce paramètre est obligatoire si vous utilisez des noms par défaut pour vos tables de métadonnées DynamoDB (table des baux, table des états de coordinateur et table des métriques de worker). Si vous avez spécifié des noms personnalisés pour ces tables, vous pouvez omettre ce paramètre. *applicationName*Remplacez-le par le nom réel de votre application KCL. L'outil l'utilise pour créer les noms de table par défaut si aucun nom personnalisé n'est fourni.

`--lease_table_name`

Ce paramètre est nécessaire si vous avez défini un nom personnalisé pour la table des baux dans votre configuration KCL. Si vous utilisez le nom de table par défaut, vous pouvez omettre ce paramètre. *leaseTableName*Remplacez-le par le nom de table personnalisé que vous avez spécifié pour votre table de location.

`--coordinator_state_table_name`

Ce paramètre est nécessaire si vous avez défini un nom personnalisé pour la table des états de coordinateur dans votre configuration KCL. Si vous utilisez le nom de table par défaut, vous pouvez omettre ce paramètre. *coordinatorStateTableName*Remplacez-le par le nom de table personnalisé que vous avez spécifié pour votre table d'état des coordinateurs.

`--worker_metrics_table_name`

Ce paramètre est nécessaire si vous avez défini un nom personnalisé pour la table des métriques de worker dans votre configuration KCL. Si vous utilisez le nom de table par défaut, vous pouvez omettre ce paramètre. *workerMetricsTableName*Remplacez-le par le nom de table personnalisé que vous avez spécifié pour votre tableau des métriques des travailleurs.

Étape 2 : redéployer le code avec la version précédente de la KCL

> **Important**
>
> Toute mention de la version 2.x dans la sortie générée par l'outil de migration de la KCL doit être interprétée comme faisant référence à la KCL version 1.x. L'exécution du script

n’effectue pas de restauration par régression complète : elle fait uniquement basculer l’algorithme d’équilibrage de charge vers celui utilisé dans la KCL version 1.x.

Après avoir exécuté l’outil de migration de la KCL pour effectuer une restauration par régression, l’un des messages suivants s’affiche :

Message 1

« Restauration par régression terminée. Votre application exécutait une fonctionnalité compatible avec la version 2x. Veuillez restaurer les fichiers binaires précédents de votre application en déployant le code avec votre version précédente de la KCL. »

Action requise : vos workers s’exécutaient dans le mode compatible avec la KCL 1.x. Redéployez le code avec la version précédente de la KCL sur vos workers.

Message 2

« Restauration par régression terminée. Votre application KCL exécutait une fonctionnalité compatible avec la version 3x. Une fonctionnalité compatible avec la version 2x a été restaurée par régression. Si vous ne constatez aucune amélioration après un court laps de temps, restaurez les fichiers binaires précédents de votre application en déployant le code avec votre version précédente de la KCL. »

Action requise : vos workers s’exécutaient dans le mode compatible avec la KCL 3.x et l’outil de migration de la KCL a fait basculer tous les workers vers le mode compatible avec la KCL 1.x. Redéployez le code avec la version précédente de la KCL sur vos workers.

Message 3

« L’application a déjà fait l’objet d’une restauration par régression. Toutes KCLv3 les ressources susceptibles d'être supprimées ont été nettoyées afin d'éviter des frais jusqu'à ce que l'application puisse être reportée avec la migration. »

Action requise : vos workers ont déjà fait l’objet d’une restauration par régression pour s’exécuter dans le mode compatible avec la KCL 1.x. Redéployez le code avec la version précédente de la KCL sur vos workers.

Restauration par progression de la KCL 3.x après une restauration par régression

Cette rubrique explique comment restaurer par progression votre application consommateur vers la KCL 3.x après une restauration par régression. Lorsque vous devez effectuer une restauration par progression, vous devez suivre un processus en deux étapes :

1. Exécuter l'outil de migration de la KCL
2. Déployer le code avec la KCL 3.x

Étape 1 : exécuter l'outil de migration de la KCL

Exécutez l'outil de migration de la KCL avec la commande suivante pour effectuer une restauration par progression vers la KCL 3.x :

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameters

`--region`

*region*Remplacez-le par votre Région AWS.

`--application_name`

Ce paramètre est obligatoire si vous utilisez des noms par défaut pour votre table des états de coordinateur. Si vous avez spécifié des noms personnalisés pour la table des états de coordinateur, vous pouvez omettre ce paramètre. *applicationName*Remplacez-le par le nom réel de votre application KCL. L'outil l'utilise pour créer les noms de table par défaut si aucun nom personnalisé n'est fourni.

`--coordinator_state_table_name`

Ce paramètre est nécessaire si vous avez défini un nom personnalisé pour la table des états de coordinateur dans votre configuration KCL. Si vous utilisez le nom de table par défaut, vous pouvez omettre ce paramètre. *coordinatorStateTableName*Remplacez-le par le nom de table personnalisé que vous avez spécifié pour votre table d'état des coordinateurs.

Une fois que vous avez exécuté l'outil de migration en mode restauration par progression, la KCL crée les ressources DynamoDB suivantes requises pour la KCL 3.x :

- Un index secondaire global sur la table des baux
- Une table des métriques de worker

Étape 2 : déployer le code avec la KCL 3.x

Après avoir exécuté l'outil de migration de la KCL pour une restauration par progression, déployez votre code avec la KCL 3.x sur vos workers. Pour terminer votre migration, consultez Step 8: Complete the migration.

Démonstration : adaptateur Kinesis DynamoDB Streams

Cette section est une démonstration d'une application Java qui utilise la bibliothèque client Amazon Kinesis et l'adaptateur Amazon DynamoDB Streams. L'application illustre un exemple de la réplication de données, où l'activité d'écriture d'une table est appliquée à une seconde table, avec le contenu des deux tables demeurant synchronisé. Pour le code source, consultez Programme complet : adaptateur DynamoDB Streams Kinesis.

Le programme exécute les tâches suivantes :

1. Crée deux tables DynamoDB nommées `KCL-Demo-src` et `KCL-Demo-dst`. Chacune de ces tables dispose d'un flux activé sur elle-même.
2. Génère une activité de mise à jour de la table source en ajoutant, mettant à jour et supprimant des éléments. Cela entraîne l'écriture des données sur le flux de la table.
3. Lit les enregistrements du flux, les reconstruit en tant que demandes DynamoDB, et applique les demandes à la table de destination.
4. Analyse les tables source et de destination afin de s'assurer que leurs contenus sont identiques.
5. Nettoie en supprimant les tables.

Ces étapes sont décrites dans les sections suivantes et l'application complète est illustrée à la fin de la procédure pas à pas.

Rubriques

- Étape 1 : créer des tables DynamoDB
- Étape 2 : générer une activité de mise à jour de la table source
- Étape 3 : traiter le flux
- Étape 4 : s'assurer que les deux tables ont un contenu identique

- Étape 5 : nettoyer
- Programme complet : adaptateur DynamoDB Streams Kinesis

Étape 1 : créer des tables DynamoDB

La première étape consiste à créer deux tables DynamoDB : une table source et une table de destination. Le `StreamViewType` sur le flux de la table source est `NEW_IMAGE`. Cela signifie que chaque fois qu'un élément est modifié dans la table, l'image « après » de l'élément est écrite dans le flux. Ainsi, le flux assure le suivi des toute l'activité d'écriture sur la table.

L'extrait de code suivant illustre le code utilisé pour la création des deux tables.

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

Étape 2 : générer une activité de mise à jour de la table source

L'étape suivante consiste à créer une activité d'écriture sur la table source. Tandis que cette activité a lieu, le flux de la table source est aussi mis à jour en quasi-temps réel.

L'application définit une classe d'assistance avec les méthodes qui appellent les actions d'API `PutItem`, `UpdateItem` et `DeleteItem` pour écrire les données. L'extrait de code suivant montre comment ces méthodes sont utilisées.

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

Étape 3 : traiter le flux

Maintenant le programme commence à traiter le flux. L'adaptateur DynamoDB Streams Kinesis agit comme une couche transparente entre la KCL et le point de terminaison DynamoDB Streams, afin que le code puisse pleinement exploiter la KCL au lieu de devoir effectuer des appels DynamoDB Streams de bas niveau. Le programme effectue les tâches suivantes :

- Il définit une classe de processeur d'enregistrements, `StreamsRecordProcessor`, avec des méthodes conformes à la définition de l'interface KCL : `initialize`, `processRecords` et `shutdown`. La méthode `processRecords` contient la logique nécessaire pour lire à partir du flux de la table source et écrire dans la table de destination.
- Il définit une fabrique de classe pour la classe de processeur d'enregistrements (`StreamsRecordProcessorFactory`). Cette action est obligatoire pour les programmes Java qui utilisent le KCL.
- Il instancie un nouveau KCL `Worker`, associé à la fabrique de classe.
- Il arrête le `Worker` lorsque le traitement des enregistrements est terminé.

Pour en savoir plus sur la définition de l'interface de la KCL, consultez Développement d'applications consommateur à l'aide de la bibliothèque client Kinesis dans le Guide du développeur Amazon Kinesis Data Streams.

L'extrait de code suivant illustre la boucle principale dans `StreamsRecordProcessor`. L'instruction `case` détermine l'action à exécuter, en fonction de l'`OperationType` qui s'affiche dans l'enregistrement de flux.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Étape 4 : s'assurer que les deux tables ont un contenu identique

À ce stade, le contenu des tables source et destination est synchronisé. L'application émet des demandes `Scan` sur les deux tables afin de vérifier que leurs contenus sont, en fait, identiques.

La classe `DemoHelper` contient une méthode `ScanTable` qui appelle l'API `Scan` de bas niveau. L'exemple suivant illustre la marche à suivre.

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
```

```
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
}
```

Étape 5 : nettoyer

Comme la démonstration est terminée, l'application supprime les tables source et destination. Consultez l'exemple de code suivant. Même après que les tables sont supprimées, leurs flux demeurent accessibles 24 heures, délai au-delà duquel ils sont automatiquement supprimés.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Programme complet : adaptateur DynamoDB Streams Kinesis

Voici le programme Java complet qui effectue les tâches décrites dans Démonstration : adaptateur Kinesis DynamoDB Streams. Lorsque vous l'exécutez, vous devez visualiser une sortie similaire à ce qui suit.

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

> **Important**
>
> Pour exécuter ce programme, assurez-vous que l'application cliente a accès à DynamoDB et à CloudWatch Amazon à l'aide de politiques. Pour de plus amples informations, veuillez consulter Politiques basées sur l'identité pour DynamoDB.

Le code source se compose de quatre fichiers `.java` :

- `StreamsAdapterDemo.java`

- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
    private static AwsCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();
```

```
    public static void main( String[] args ) throws Exception {
        System.out.println("Starting demo...");
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        amazonDynamoDbStreamsAdapterClient = new
AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);

        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

        setUpTables();

        StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

        ShardRecordProcessorFactory shardRecordProcessorFactory =
                new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                "streams-adapter-demo",
                amazonDynamoDbStreamsAdapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                "streams-demo-worker",
                shardRecordProcessorFactory
        );

        DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
        RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
        retrievalConfig.retrievalSpecificConfig(pollingConfig);

        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
```

```
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();

        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
```

```
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
"test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
"test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
"demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
"demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());

dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
```

```
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
 tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.checkpointCounter = 0;
    }

    @Override
```

```
    public void processRecords(DynamoDBStreamsProcessRecordsInput
dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
```

```
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
```

```
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();

        try {
            System.out.println("Creating table " + tableName);
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
        return dynamoDbClient.scan(ScanRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Id", AttributeValue.builder().n(id).build());
        item.put("attribute-1", AttributeValue.builder().s(val).build());

        putItem(dynamoDbClient, tableName, item);
    }
```

```
    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                               Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(items)
                .build();
        dynamoDbClient.putItem(putItemRequest).join();
    }

    public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id, String val) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attr2", "attribute-2");

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

        UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #attr2 = :val")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

API de bas niveau DynamoDB Streams : exemple Java

> **Note**
>
> Le code sur cette page n'est pas exhaustif et ne gère pas tous les scénarios de consommation d'Amazon DynamoDB Streams. La manière recommandée d'utiliser les enregistrements de flux de DynamoDB est de le faire via l'adaptateur Amazon Kinesis en utilisant la bibliothèque client Kinesis (KCL), comme décrit dans Utilisation de l'adaptateur DynamoDB Streams Kinesis pour traiter des enregistrements de flux.

Cette section contient un programme Java qui affiche DynamoDB Streams en action. Le programme exécute les tâches suivantes :

1. Crée une table DynamoDB avec un flux activé.
2. Décrit les paramètres de flux de cette table.
3. Modifie les données de la table.
4. Décrit les partitions du flux.
5. Lit les enregistrements de flux des partitions.
6. Récupère les partitions enfants et continue de lire les enregistrements.
7. Nettoie.

Lorsque vous exécutez le programme, vous obtenez une sortie similaire à ce qui suit :

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
```

```
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 100002572486797508907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583390 9, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789 2701, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=220000109977111289 8434, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

Example Exemple

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
```

```
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }

            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }

            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
```

```
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }

    private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
        for (Shard shard : shards) {
            String shardId = shard.shardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard
            GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
                    .streamArn(streamArn).shardId(shardId)
                    .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();
```

```
            GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

            String currentShardIter = getShardIteratorResult.shardIterator();

            // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
            // To prevent running the loop until the Shard is sealed, we process only
the
            // items that were written into DynamoDB and then exit.
            int processedRecordCount = 0;
            while (currentShardIter != null && processedRecordCount < maxItemCount) {
                // Use the shard iterator to read the stream records
                GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                        .shardIterator(currentShardIter).build();
                GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
                List<Record> records = getRecordsResult.records();
                for (Record record : records) {
                    System.out.println("        " + record.dynamodb());
                }
                processedRecordCount += records.size();
                currentShardIter = getRecordsResult.nextShardIterator();
            }
            if (currentShardIter == null){
                System.out.println("Shard has been fully processed. Shard iterator is
null.");
                System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
                DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
        }
    }
```

```
   private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
       Map<String, AttributeValue> item = new HashMap<>();
       item.put(key, AttributeValue.builder()
               .s(i.toString())
               .build());
       item.put("Message", AttributeValue.builder()
               .s("New Item!")
               .build());
       PutItemRequest request = PutItemRequest.builder()
               .tableName(tableName)
               .item(item)
               .build();
       ddb.putItem(request);
   }

   private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

       HashMap<String, AttributeValue> itemKey = new HashMap<>();
       itemKey.put(key, AttributeValue.builder()
               .s(i.toString())
               .build());


       HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
       updatedValues.put("Message", AttributeValueUpdate.builder()
               .value(AttributeValue.builder().s("This is an updated item").build())
               .action(AttributeAction.PUT)
               .build());

       UpdateItemRequest request = UpdateItemRequest.builder()
               .tableName(tableName)
               .key(itemKey)
               .attributeUpdates(updatedValues)
               .build();
       ddb.updateItem(request);
   }

   public static void deleteDynamoDBItem(String key, Integer i, String tableName,
DynamoDbClient ddb) {
       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put(key, AttributeValue.builder()
               .s(i.toString())
```

```
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
    }

    public static TableDescription createTable(DynamoDbClient ddb, String tableName,
String key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType("NEW_AND_OLD_IMAGES")
                .build();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
                .tableName(tableName)
                .streamSpecification(streamSpecification)
                .build();

        TableDescription newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            System.out.println("Waiting for " + tableName + " to be created...");

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
```

```
            newTable = response.tableDescription();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return null;
    }



}
```

Streams et déclencheurs DynamoDB AWS Lambda

Amazon DynamoDB est intégré afin que vous puissiez créer AWS Lambda des déclencheurs, des éléments de code qui répondent automatiquement aux événements dans DynamoDB Streams. Avec des déclencheurs, vous pouvez créer des applications qui réagissent à des modifications de données dans des tables DynamoDB.

Rubriques

- Tutoriel #1 : Utilisation de filtres pour traiter tous les événements avec Amazon AWS Lambda DynamoDB et utilisation du AWS CLI
- Tutoriel n° 2 : utilisation de filtres pour traiter certains événements avec DynamoDB et Lambda
- Bonnes pratiques relatives à l'utilisation de DynamoDB Streams avec Lambda

Si vous activez DynamoDB Streams sur une table, vous pouvez associer le flux Amazon Resource Name (ARN) à une AWS Lambda fonction que vous écrivez. Toutes les actions de mutation vers cette table DynamoDB peuvent ensuite être capturées en tant qu'élément dans le flux. Par exemple, vous pouvez définir un déclencheur pour que, lorsqu'un élément de table est modifié, un nouvel enregistrement apparaisse immédiatement dans le flux de cette table.

> **Note**
>
> Si vous abonnez plus de deux fonctions Lambda à un flux DynamoDB, une limitation de lecture peut se produire.

Le service AWS Lambda interroge le flux à la recherche de nouveaux enregistrements quatre fois par seconde. Lorsque de nouveaux enregistrements de flux sont disponibles, votre fonction Lambda est invoquée de manière synchrone. Vous pouvez abonner jusqu'à deux fonctions Lambda au même flux DynamoDB. Si vous abonnez plus de deux fonctions Lambda au même flux DynamoDB, une limitation de lecture peut se produire.

La fonction Lambda peut envoyer une notification, lancer un flux de travail ou effectuer un grand nombre d'actions spécifiées par vos soins. Vous pouvez écrire une fonction Lambda simplement afin de copier chaque enregistrement de flux vers un stockage permanent tel qu'Amazon S3 File Gateway (Amazon S3) et créer une piste d'audit permanente de l'activité d'écriture dans votre table. Ou supposons que vous ayez une application de jeux pour appareils mobiles qui écrive dans une table `GameScores`. Chaque fois que l'attribut `TopScore` de la table `GameScores` est mis à jour, un enregistrement de flux correspondant est écrit dans le flux de la table. Cet événement peut alors déclencher une fonction Lambda qui publie un message de félicitations sur un réseau social. Cette fonction ignorerait simplement tout enregistrement de flux qui ne serait pas une mise à jour de `GameScores`, ou qui ne modifierait pas l'attribut `TopScore`.

Si votre fonction renvoie une erreur, Lambda réessaie de traiter le lot jusqu'à ce que le traitement réussisse ou que les données expirent. Vous pouvez également configurer Lambda pour effectuer de nouvelles tentatives avec un lot plus petit, limiter le nombre de tentatives, supprimer les enregistrements une fois qu'ils sont trop anciens et d'autres options.

Afin de respecter les bonnes pratiques en matière de performances, la fonction Lambda doit être de courte durée. Pour éviter d'introduire des retards de traitement inutiles, elle ne doit pas non plus exécuter de logique complexe. Pour un flux à haute vitesse en particulier, il est préférable de déclencher des flux de travail asynchrones avec des fonctions de post-traitement par étapes plutôt que des Lambdas synchrones de longue durée.

Vous ne pouvez pas utiliser le même déclencheur Lambda sur différents comptes.AWS La table DynamoDB et les fonctions Lambda doivent appartenir au même compte.AWS

Pour plus d'informations AWS Lambda, consultez le guide du AWS Lambda développeur.

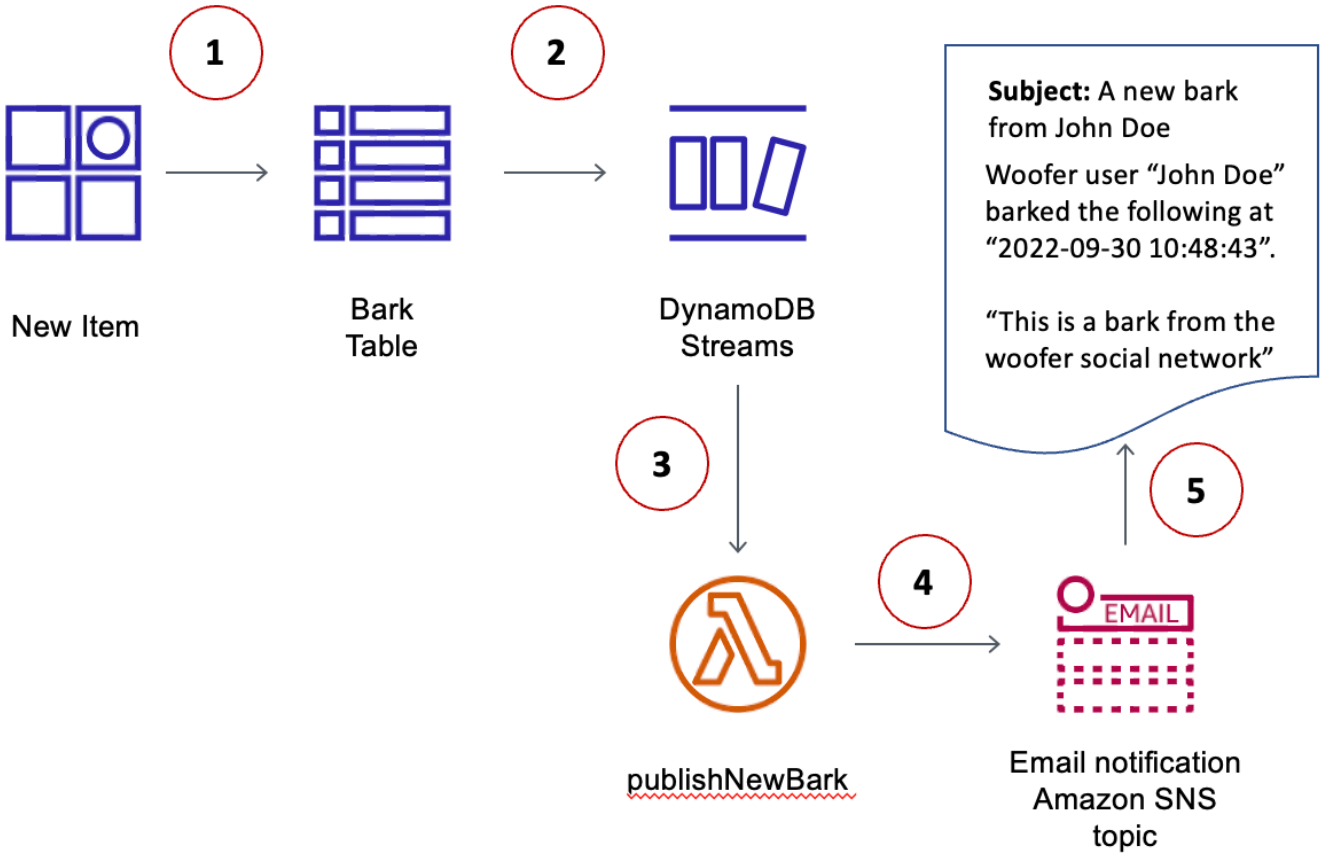
Tutoriel #1 : Utilisation de filtres pour traiter tous les événements avec Amazon AWS Lambda DynamoDB et utilisation du AWS CLI

Dans ce didacticiel, vous allez créer un AWS Lambda déclencheur pour traiter un flux provenant d'une table DynamoDB.

Rubriques

- Étape 1 : créer une table DynamoDB avec un flux activé
- Étape 2 : créer un rôle d'exécution Lambda
- Étape 3 : créer une rubrique Amazon SNS
- Étape 4 : créer et tester une fonction Lambda
- Étape 5 : créer et tester un déclencheur

Le scénario de ce didacticiel est Woofer, un réseau social simple. Les utilisateurs de Woofer communiquent avec des aboiements (messages textuels brefs) qui sont envoyés à d'autres utilisateurs de Woofer. Le schéma suivant illustre les composants et le flux de travail de cette application.



1. Un utilisateur écrit un élément dans une table DynamoDB (`BarkTable`). Chaque élément de la table représente un aboiement.
2. Un nouvel enregistrement de flux est écrit pour refléter l'ajout de ce nouvel élément à `BarkTable`.
3. Le nouvel enregistrement du flux déclenche une AWS Lambda fonction (`publishNewBark`).

4. Si l'enregistrement de flux indique qu'un nouvel élément a été ajouté à `BarkTable`, la fonction Lambda lit les données de l'enregistrement de flux et publie un message sur une rubrique dans Amazon Simple Notification Service (Amazon SNS).
5. Le message est reçu par les abonnés à la rubrique Amazon SNS. Dans le cadre de ce didacticiel, le seul abonné est une adresse e-mail.

Avant de commencer

Ce didacticiel utilise le AWS Command Line InterfaceAWS CLI. Si vous ne l'avez déjà fait, suivez les instructions du Guide de l'utilisateur AWS Command Line Interface pour installer et configurer l'AWS CLI.

Étape 1 : créer une table DynamoDB avec un flux activé

Au cours de cette étape, vous allez créer une table DynamoDB (`BarkTable`) pour stocker tous les aboiements des utilisateurs de Woofer. La clé primaire se compose de `Username` (clé de partition) et de `Timestamp` (clé de tri). Les deux attributs sont de type string (chaîne).

`BarkTable` a un flux activé. Plus loin dans ce didacticiel, vous allez créer un déclencheur en associant une AWS Lambda fonction au flux.

1. Créez le flux à l'aide de la commande suivante.

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. Dans la sortie, recherchez le `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Notez *`region`* et *`accountID`*, car vous en aurez besoin pour les autres étapes de ce didacticiel.

Étape 2 : créer un rôle d'exécution Lambda

Au cours de cette étape, vous créez un rôle Gestion des identités et des accès AWS(IAM) (`WooferLambdaRole`) et vous lui attribuez des autorisations. Ce rôle sera utilisé par la fonction Lambda que vous allez créer dans Étape 4 : créer et tester une fonction Lambda.

Vous allez également créer une politique pour le rôle. Cette politique contient toutes les autorisations dont la fonction Lambda a besoin lors de l'exécution.

1. Créez un fichier nommé `trust-relationship.json` avec les contenus suivants.

   JSON

   ```
   {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
   ```

2. Entrez la commande suivante pour créer `WooferLambdaRole`.

   ```
   aws iam create-role --role-name WooferLambdaRole \
       --path "/service-role/" \
       --assume-role-policy-document file://trust-relationship.json
   ```

3. Créez un fichier nommé `role-policy.json` avec les contenus suivants. (Remplacez *`region`* et *`accountID`* par votre AWS région et votre numéro de compte.)

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
   ```

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

La politique comporte quatre instructions pour autoriser `WooferLambdaRole` à effectuer les opérations suivantes :

- Exécutez une fonction Lambda (`publishNewBark`). Vous allez créer cette fonction plus tard dans le cadre de ce didacticiel.
- Accédez à Amazon CloudWatch Logs. La fonction Lambda écrit les diagnostics dans les CloudWatch journaux au moment de l'exécution.
- Lisez les données du flux DynamoDB pour `BarkTable`.
- Publiez des messages sur Amazon SNS.

4. Exécutez la commande suivante pour associer la politique à `WooferLambdaRole`.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

Étape 3 : créer une rubrique Amazon SNS

Au cours de cette étape, vous allez créer une rubrique Amazon SNS (`wooferTopic`) et y abonner une adresse e-mail. Votre fonction Lambda utilise cette rubrique pour publier de nouveaux aboiements des utilisateurs de Woofer.

1. Entrez la commande suivante pour créer une rubrique Amazon SNS.

```
aws sns create-topic --name wooferTopic
```

2. Tapez la commande suivante pour abonner une adresse e-mail à `wooferTopic`. (Remplacez *`region`* et *`accountID`* par votre région AWS et votre ID de compte, puis remplacez *`example@example.com`* par une adresse e-mail valide.)

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS envoie un message de confirmation à votre adresse e-mail. Cliquez sur le lien Confirm subscription (Confirmer l'abonnement) de ce message pour finaliser le processus d'abonnement.

Étape 4 : créer et tester une fonction Lambda

Au cours de cette étape, vous créez une AWS Lambda fonction (`publishNewBark`) pour traiter les enregistrements de flux à partir de`BarkTable`.

La fonction `publishNewBark` traite uniquement les événements de flux qui correspondent aux nouveaux éléments de `BarkTable`. La fonction lit les données d'un tel événement, puis appelle Amazon SNS pour les publier.

1. Créez un fichier nommé `publishNewBark.js` avec les contenus suivants. Remplacez *`region`* et *`accountID`* par votre AWS région et votre numéro de compte.

```
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. Créez un fichier zip pour `publishNewBark.js`. Si vous disposez de l'utilitaire de ligne de commande pour zipper, vous pouvez taper la commande suivante.

```
zip publishNewBark.zip publishNewBark.js
```

3. Lorsque vous créez la fonction Lambda, vous spécifiez l'Amazon Resource Name (ARN) pour `WooferLambdaRole`, que vous avez créé dans Étape 2 : créer un rôle d'exécution Lambda. Tapez la commande suivante pour extraire cet ARN.

```
aws iam get-role --role-name WooferLambdaRole
```

Dans la sortie, recherchez l'ARN pour `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Utilisez la commande suivante pour créer la fonction Lambda. Remplacez *roleARN* par l'ARN pour`WooferLambdaRole`.

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. Maintenant, testez `publishNewBark` pour vérifier le bon fonctionnement. Pour cela, vous entrez des informations similaires à un enregistrement réel de DynamoDB Streams.

Créez un fichier nommé `payload.json` avec les contenus suivants. Remplacez *region* et *accountID* par votre Région AWS et votre ID de compte.

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
```

```
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "130216000000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

Testez la fonction `publishNewBark` à l'aide de la commande suivante.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Si le test est réussi, vous verrez la sortie suivante.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

De plus, le fichier `output.txt` contiendra le texte suivant.

```
"Successfully processed 1 records."
```

Vous recevrez également un nouveau message électronique quelques minutes après.

> **Note**
>
> AWS Lambdaécrit des informations de diagnostic dans Amazon CloudWatch Logs. Si vous rencontrez des erreurs avec la fonction Lambda, vous pouvez utiliser ces diagnostics à des fins de dépannage :
>
> 1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
> 2. Dans le panneau de navigation, sélectionnez Logs (Journaux).
> 3. Choisissez le groupe de journal suivant : `/aws/lambda/publishNewBark`
> 4. Choisissez le dernier flux de journal pour consulter la sortie (et les erreurs) de la fonction.

Étape 5 : créer et tester un déclencheur

Dans Étape 4 : créer et tester une fonction Lambda, vous avez testé la fonction Lambda pour vérifier qu'elle s'exécutait correctement. Au cours de cette étape, vous allez créer un déclencheur en associant la fonction Lambda (`publishNewBark`) à une source d'événement (le flux `BarkTable`).

1. Lorsque vous créez le déclencheur, vous devez spécifier l'ARN du flux `BarkTable`. Tapez la commande suivante pour extraire cet ARN.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Dans la sortie, recherchez le `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Tapez la commande suivante pour créer le déclencheur. Remplacez *streamARN* par l'ARN du flux réel.

```
aws lambda create-event-source-mapping \
    --region region \
    --function-name publishNewBark \
    --event-source streamARN  \
    --batch-size 1 \
    --starting-position TRIM_HORIZON
```

3. Testez le déclencheur. Tapez la commande suivante pour ajouter un élément à `BarkTable`.

```
aws dynamodb put-item \
    --table-name BarkTable \
    --item Username={S="Jane
 Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
```

   Vous devriez recevoir un nouveau message électronique quelques minutes après.

4. Ouvrez la console DynamoDB et ajoutez quelques éléments à `BarkTable`. Vous devez spécifier les valeurs des attributs `Username` et `Timestamp`. Vous devez également spécifier une valeur pour `Message`, bien que cela ne soit pas obligatoire. Vous devriez recevoir un nouveau message électronique pour chaque élément ajouté à `BarkTable`.

   La fonction Lambda traite uniquement les nouveaux éléments que vous ajoutez à `BarkTable`. Si vous mettez à jour ou supprimez un élément dans la table, la fonction ne fait rien.

> **Note**
>
> AWS Lambdaécrit des informations de diagnostic dans Amazon CloudWatch Logs. Si vous rencontrez des erreurs avec la fonction Lambda, vous pouvez utiliser ces diagnostics à des fins de dépannage.
>
> 1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
> 2. Dans le panneau de navigation, sélectionnez Logs (Journaux).
> 3. Choisissez le groupe de journal suivant : `/aws/lambda/publishNewBark`
> 4. Choisissez le dernier flux de journal pour consulter la sortie (et les erreurs) de la fonction.

Tutoriel n° 2 : utilisation de filtres pour traiter certains événements avec DynamoDB et Lambda

Dans ce didacticiel, vous allez créer un AWS Lambda déclencheur pour traiter uniquement certains événements d'un flux à partir d'une table DynamoDB.

Rubriques

- Synthèse –CloudFormation
- Synthèse – CDK

Avec le filtrage des événements Lambda, vous pouvez utiliser des expressions de filtre pour contrôler les événements envoyés par Lambda à votre fonction pour traitement. Vous pouvez configurer jusqu'à 5 filtres différents par flux DynamoDB. Si vous utilisez des fenêtres de traitement par lots, Lambda applique les critères de filtre à chaque nouvel événement pour déterminer s'il doit être ajouté au lot actuel.

Les filtres sont appliqués via des structures nommées `FilterCriteria`. Les 3 principaux attributs de `FilterCriteria` sont `metadata properties`, `data properties` et `filter patterns`.

Voici un exemple de structure d'un événement DynamoDB Streams :

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
```

```
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Les `metadata properties` correspondent aux champs de l'objet d'événement. Dans le cas de DynamoDB Streams, les `metadata properties` correspondent à des champs tels que `dynamodb` ou `eventName`.

Les `data properties` correspondent aux champs du corps de l'événement. Pour filtrer sur `data properties`, veillez à les contenir dans `FilterCriteria` à l'intérieur de la clé appropriée. Pour les sources d'événements DynamoDB, la clé de données est `NewImage` ou `OldImage`.

En dernier lieu, les règles de filtrage définissent l'expression de filtre que vous appliquez à une propriété spécifique. Voici quelques exemples :

Opérateur de comparaison	Exemple	Syntaxe des règles (partielle)
Null	Type de produit null	`{ "product_type": { "S": null } }`
Vide	Nom du produit vide	`{ "product_name": { "S": [ ""] } }`
Égal à	État égal à Floride	`{ "state": { "S": ["FL"] } }`
Et	État du produit égal à Floride et catégorie de produit Chocolat	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
Or	L'État du produit est la Floride ou la Californie	`{ "state": { "S": ["FL","CA"] } }`

Opérateur de comparaison	Exemple	Syntaxe des règles (partielle)
Pas	L'État du produit n'est pas la Floride	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Existe	Le produit fait maison existe	`{"homemade": {"S": [{"exists": true}]}}`
N'existe pas	Le produit fait maison n'existe pas	`{"homemade": {"S": [{"exists": false}]}}`
Commence par	PK commence par COMPANY	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Vous pouvez spécifier jusqu'à 5 modèles de filtrage des événements pour une fonction Lambda. Notez que chacun de ces 5 événements sera évalué comme un OR logique. Donc, si vous configurez deux filtres nommés `Filter_One` et `Filter_Two`, la fonction Lambda exécutera `Filter_One` OR `Filter_Two`.

> **Note**
>
> La page de filtrage des événements Lambda propose certaines options permettant de filtrer et de comparer des valeurs numériques. Toutefois, dans le cas des événements de filtre DynamoDB, cela ne s'applique pas, car les nombres dans DynamoDB sont stockés sous forme de chaînes. Par exemple `"quantity": { "N": "50" }`, nous savons que c'est un numéro grâce à la propriété `"N"`.

Synthèse –CloudFormation

Pour illustrer la fonctionnalité de filtrage des événements dans la pratique, voici un exemple de CloudFormation modèle. Ce modèle va générer une table DynamoDB simple avec une clé de partition PK et une clé de tri SK avec Amazon DynamoDB Streams activé. Il créera une fonction Lambda et un rôle d'exécution Lambda simple qui permettront d'écrire des journaux dans Amazon Cloudwatch et de lire les événements depuis le flux Amazon DynamoDB. Il ajoutera également le mappage des sources d'événements entre les DynamoDB Streams et la fonction Lambda, afin que

la fonction puisse être exécutée chaque fois qu'un événement se produit dans Amazon DynamoDB Stream.

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
          KeyType: "HASH"
        - AttributeName: "SK"
          KeyType: "RANGE"
      StreamSpecification:
        StreamViewType: "NEW_AND_OLD_IMAGES"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17",
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
```

```
        PolicyDocument:
          Version: "2012-10-17",
          Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource: arn:aws:logs:*:*:*
            - Effect: Allow
              Action:
                - dynamodb:DescribeStream
                - dynamodb:GetRecords
                - dynamodb:GetShardIterator
                - dynamodb:ListStreams
              Resource: !GetAtt StreamsSampleDDBTable.StreamArn

  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
              LOGGER.info('Record: %s', rec)
```

```
Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

Après avoir déployé ce modèle CloudFormation, vous pouvez insérer l'élément Amazon DynamoDB suivant :

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Grâce à la fonction lambda simple incluse en ligne dans ce modèle de formation de cloud, vous verrez les événements relatifs à la fonction lambda dans les groupes de CloudWatch logs Amazon comme suit :

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
```

```
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Exemples de filtrage

- Uniquement les produits qui correspondent à un État donné

Cet exemple modifie le CloudFormation modèle pour inclure un filtre correspondant à tous les produits provenant de Floride, avec l'abréviation « FL ».

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Une fois que vous avez redéployé la pile, vous pouvez ajouter l’élément DynamoDB suivant à la table. Notez qu’il n’apparaîtra pas dans les journaux des fonctions Lambda, car le produit de cet exemple provient de Californie.

```
{
```

```
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- Uniquement les éléments qui commencent par certaines valeurs dans PK et SK

Cet exemple modifie le CloudFormation modèle pour inclure la condition suivante :

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
 "COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Notez que la condition AND exige que la condition se trouve à l'intérieur du modèle, où les clés PK et SK se trouvent dans la même expression, séparées par des virgules.

Soit vous commencez avec des valeurs sur PK et SK, soit le produit provient d'un État donné.

Cet exemple modifie le CloudFormation modèle pour inclure les conditions suivantes :

```
  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
```

```
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Notez que la condition OR est ajoutée en introduisant de nouveaux modèles dans la section des filtres.

Synthèse – CDK

L'exemple de modèle de formation de projet CDK suivant décrit les fonctionnalités de filtrage des événements. Avant de travailler avec ce projet CDK, vous devez installer les prérequis, y compris l'exécution de scripts de préparation.

Créer un projet CDK

Créez d'abord un nouveau AWS CDK projet en l'invoquant `cdk init` dans un répertoire vide.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

La commande `cdk init` utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Tous les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Dans le cas contraire, le nom doit prendre la forme d'un identifiant Python. Par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Pour travailler avec le nouveau projet, activez son environnement virtuel. Cela permet d'installer les dépendances du projet localement dans le dossier du projet, plutôt que globalement.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> Il se peut que vous reconnaissiez qu'il s'agit de la Mac/Linux commande permettant d'activer un environnement virtuel. Les modèles Python incluent un fichier batch, `source.bat`, qui

permet d'utiliser la même commande sous Windows. La commande Windows traditionnelle `.venv\Scripts\activate.bat` fonctionne également. Si vous avez initialisé votre AWS CDK projet à l'aide de AWS CDK Toolkit v1.70.0 ou d'une version antérieure, votre environnement virtuel se trouve dans le `.env` répertoire au lieu de. `.venv`

Infrastructure de base

Ouvrez le fichier `./ddb_filters/ddb_filters_stack.py` dans l'éditeur de texte de votre choix. Ce fichier a été généré automatiquement lorsque vous avez créé le AWS CDK projet.

Ensuite, ajoutez les fonctions `_create_ddb_table` et `_set_ddb_trigger_function`. Ces fonctions créeront une table DynamoDB avec la clé de partition PK et la clé de tri SK en mode provisionné et mode à la demande), avec Amazon DynamoDB Streams activé par défaut pour afficher les nouvelles et les anciennes images.

La fonction Lambda sera stockée dans le dossier `lambda` situé sous le fichier `app.py`. Ce fichier sera créé ultérieurement. Elle inclura une variable d'environnement `APP_TABLE_NAME`, qui sera le nom de la table Amazon DynamoDB créée par cette pile. Dans la même fonction, nous accorderons des autorisations de lecture de flux à la fonction Lambda. Enfin, elle s'abonnera aux DynamoDB Streams en tant que source d'événements pour la fonction Lambda.

À la fin du fichier de la méthode `__init__`, vous appellerez les constructions respectives pour les initialiser dans la pile. Pour les projets plus importants qui nécessitent des composants et des services supplémentaires, il peut être préférable de définir ces constructions en dehors de la pile de base.

```
import os
import json

import aws_cdk as cdk
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):
```

```
    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )

        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
```

```
        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

Nous allons maintenant créer une fonction lambda très simple qui imprimera les journaux sur Amazon CloudWatch. Pour ce faire, créez un nouveau dossier appelé `lambda`.

```
mkdir lambda
touch app.py
```

À l'aide de votre éditeur de texte préféré, ajoutez le contenu suivant au fichier `app.py` :

```
import logging

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
        LOGGER.info('Record: %s', rec)
```

Assurez-vous que vous vous trouvez dans le dossier `/ddb_filters/`, tapez la commande suivante pour créer l'exemple d'application :

```
cdk deploy
```

à un moment donné, il vous sera demandé de confirmer si vous souhaitez déployer la solution. Acceptez les modifications en saisissant Y.

```
###############################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
###############################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s
```

```
Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

Une fois les modifications déployées, ouvrez votre AWS console et ajoutez un élément à votre tableau.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Les CloudWatch journaux devraient désormais contenir toutes les informations de cette entrée.

Exemples de filtrage

- Uniquement les produits qui correspondent à un État donné

Ouvrez le fichier `ddb_filters/ddb_filters/ddb_filters_stack.py` et modifiez-le pour inclure le filtre correspondant à tous les produits égaux à « FL ». Cela peut être révisé juste sous `event_subscription`, à la ligne 45.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
```

```
    },
)
```

- Uniquement les éléments qui commencent par certaines valeurs dans PK et SK

Modifiez le script Python de façon à inclure la condition suivante :

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- Soit vous commencez avec des valeurs sur PK et SK, soit le produit provient d'un État donné.

Modifiez le script Python de façon à inclure les conditions suivantes :

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
```

```
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

Notez que la condition OR est ajoutée en ajoutant d'autres éléments au tableau de filtres.

Nettoyage

Localisez la pile de filtres dans la base de votre répertoire de travail et exécutez `cdk destroy`. Il vous sera demandé de confirmer la suppression de la ressource :

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Bonnes pratiques relatives à l'utilisation de DynamoDB Streams avec Lambda

Une AWS Lambda fonction s'exécute dans un conteneur, c'est-à-dire un environnement d'exécution isolé des autres fonctions. Lorsque vous exécutez une fonction pour la première fois,AWS Lambda créez un nouveau conteneur et commencez à exécuter le code de la fonction.

Une fonction Lambda a un gestionnaire qui est exécuté une fois par appel. Le gestionnaire contient la logique métier principale de la fonction. Par exemple, la fonction Lambda illustrée dans Étape 4 : créer et tester une fonction Lambda comporte un gestionnaire qui peut traiter des enregistrements dans un flux DynamoDB.

Vous pouvez également fournir un code d'initialisation qui ne s'exécute qu'une seule fois, après la création du conteneur, mais avant la première AWS Lambda exécution du gestionnaire. La fonction

Lambda présentée dans Étape 4 : créer et tester une fonction Lambda contient un code d'initialisation qui importe le SDK JavaScript dans Node.js et crée un client pour Amazon SNS. Ces objets doivent être définis une seule fois, en dehors du gestionnaire.

Une fois la fonction exécutée, vous AWS Lambda pouvez choisir de réutiliser le conteneur pour les appels ultérieurs de la fonction. Dans ce cas, votre gestionnaire de fonction peut réutiliser les ressources que vous avez définies dans votre code d'initialisation. (Notez que vous ne pouvez pas contrôler la durée de conservation du conteneur par AWS Lambda, ni la réutilisation du conteneur.)

Pour AWS Lambda utiliser des déclencheurs DynamoDB, nous recommandons ce qui suit :

- AWS les clients de service doivent être instanciés dans le code d'initialisation, et non dans le gestionnaire. Cela permet AWS Lambda de réutiliser les connexions existantes, pendant toute la durée de vie du conteneur.
- En général, il n'est pas nécessaire de gérer explicitement les connexions ou d'implémenter un regroupement de connexions, car il AWS Lambda gère cela pour vous.

Un consommateur Lambda d'un flux DynamoDB ne garantit pas une livraison unique et peut entraîner des doublons occasionnels. Assurez-vous que le code de votre fonction Lambda est idempotent, afin d'éviter que des problèmes inattendus ne surviennent en raison d'un traitement dupliqué.

Pour plus d'informations, consultez la section Meilleures pratiques d'utilisation des AWS Lambda fonctions dans le Guide du AWS Lambda développeur.

DynamoDB Streams et Apache Flink

Vous pouvez utiliser des enregistrements Amazon DynamoDB Streams avec Apache Flink. Grâce au service géré Amazon pour Apache Flink, vous pouvez transformer et analyser des données de streaming en temps réel à l'aide d'Apache Flink. Apache Flink est un cadre de traitement de flux open source dédié au traitement de données en temps réel. Le connecteur Amazon DynamoDB Streams pour Apache Flink simplifie la création et la gestion des charges de travail Apache Flink et vous permet d'intégrer des applications à d'autres Services AWS.

Amazon Managed Service pour Apache Flink vous aide à créer rapidement des applications de traitement de end-to-end flux pour l'analyse des journaux, l'analyse des flux de clics, l'Internet des objets (IoT), les technologies publicitaires, les jeux vidéo, etc. Les quatre cas d'utilisation les plus courants sont le streaming extract-transform-load (ETL), les applications pilotées par les événements, les analyses réactives en temps réel et les requêtes interactives des flux de données. Pour plus

d’informations sur l’écriture dans Apache Flink à partir d’Amazon DynamoDB, consultez Amazon DynamoDB Streams Connector.

Utilisation du ShardFilter paramètre avec le connecteur DynamoDB Streams pour Apache Flink

Amazon DynamoDB Streams prend en charge le paramètre `ShardFilter` dans l’API `DescribeStream` pour rechercher un sous-ensemble de partitions. En spécifiant une partition parent dans la demande, DynamoDB Streams renvoie ses partitions enfants immédiates. Le paramètre `ShardFilter` vous permet de découvrir rapidement les partitions enfants après la fermeture d’une partition parent, afin de rendre vos applications de traitement de flux plus réactives et plus économiques.

Le service géré Amazon pour Apache Flink prend en charge le paramètre `ShardFilter` en cas de lecture depuis DynamoDB Streams. Pour utiliser cette fonctionnalité, vous devez demander l’accès à une version spéciale du connecteur Flink qui inclut la prise en charge du paramètre `ShardFilter`. Pour commencer, ouvrez un dossier d'assistance dans le Centre de AWS support et demandez l'accès à ces fichiers binaires. Pour plus d’informations sur l’utilisation de DynamoDB Streams avec Apache Flink, consultez Lecture et traitement de flux dans le Guide du développeur DynamoDB.

Accélération en mémoire avec DynamoDB Accelerator (DAX)

Amazon DynamoDB est conçu pour la mise à l'échelle et les performances. Dans la plupart des cas, les temps de réponse de DynamoDB sont de l'ordre de quelques millisecondes. Cependant, certains cas d'utilisation requièrent des temps de réponse de l'ordre de quelques microsecondes. Pour ces cas d'utilisation, DynamoDB Accelerator (DAX) offre des temps de réponse rapides pour accéder aux données éventuellement cohérentes.

DAX est un service de mise en cache compatible avec DynamoDB, qui vous permet de bénéficier de performances en mémoire rapides pour les applications exigeantes. DAX convient pour trois scénarios principaux :

1. En tant que cache en mémoire, DAX réduit les temps de réponse des charges de travail de lecture éventuellement cohérentes de manière très significative, de quelques millisecondes à quelques microsecondes.
2. DAX réduit la complexité opérationnelle et applicative en fournissant un service géré compatible API avec DynamoDB. Par conséquent il nécessite uniquement des modifications fonctionnelles minimales pour une utilisation avec une application existante.
3. Pour des charges de travail de lecture intensive ou en rafales, DAX offre un débit accru et peut permettre de réaliser des économies de coûts opérationnels en réduisant la nécessité de sur-approvisionner les unités de capacité de lecture. Cela s'avère particulièrement avantageux pour les applications qui nécessitent plusieurs lectures pour des clés individuelles.

DAX prend en charge le chiffrement côté serveur. Avec le chiffrement au repos, les données que DAX conserve sur disque sont chiffrées. DAX écrit les données sur disque dans le cadre de la propagation des modifications du nœud principal aux nœuds de réplica en lecture. Pour de plus amples informations, veuillez consulter Chiffrement au repos DAX.

DAX prend également en charge le chiffrement en transit garantissant que toutes les demandes et réponses entre votre application et le cluster sont chiffrées par TLS (Transport Level Security), et que les connexions au cluster peuvent être authentifiées par vérification d'un certificat x509 de cluster. Pour de plus amples informations, veuillez consulter Chiffrement DAX en transit.

Rubriques

- Cas d'utilisation pour DAX

- Notes d'utilisation de DAX
- Fonctionnement de DAX
- Composants de cluster DAX
- Création d'un cluster DAX
- Modèles de cohérence DAX et DynamoDB
- Développement avec le client DynamoDB Accelerator (DAX)
- Gestion des clusters DAX
- Surveillance de l'accélérateur DynamoDB
- Instances extensibles DAX T3/T2
- Contrôle d'accès à DAX
- Chiffrement au repos DAX
- Chiffrement DAX en transit
- Utilisation des rôles IAM liés à un service pour DAX
- Accès au DAX sur plusieurs comptes AWS
- Guide de dimensionnement de cluster DAX

Cas d'utilisation pour DAX

DAX donne accès à des données éventuellement cohérentes issues de tables DAX, avec une latence de l'ordre de quelques microsecondes. Un cluster DAX Multi-AZ peut traiter plusieurs millions de demandes par seconde.

DAX est une solution idéale pour les types d'applications suivants :

- Les applications qui ont besoin du temps de réponse le plus rapide possible pour les opérations de lecture. Ce peut être le cas, par exemple, d'applications d'enchères en temps réel, de jeux sociaux et d'opérations boursières. DAX offre des performances de lecture en mémoire élevées pour ces cas d'utilisation.
- Les applications qui lisent un petit nombre d'éléments plus souvent que d'autres. Prenons l'exemple d'un système d'e-commerce qui propose une vente d'un jour pour un produit à succès. En cours de vente, la demande de ce produit (et ses données dans DynamoDB) augmente brusquement par rapport à celle de tous les autres produits. Pour atténuer les effets de la forte

activité et d'une distribution du trafic non uniforme, vous pouvez décharger l'activité de lecture sur un cache DAX, jusqu'à ce que la vente d'un jour se termine.

- Les applications qui demandent beaucoup d'opérations de lecture, mais qui sont également sensibles aux coûts. Avec DynamoDB, vous approvisionnez le nombre d'opération de lecture par seconde dont votre application a besoin. Si l'activité de lecture s'accroît, vous pouvez augmenter le débit de lecture alloué de vos tables (moyennant un coût supplémentaire). Vous pouvez également décharger l'activité de votre application sur un cluster DAX et réduire la quantité d'unités de capacité de lecture que vous devriez autrement acheter.
- Les applications qui imposent des opérations de lecture à répétition dans un jeu de données volumineux. Une application de ce type peut potentiellement détourner les ressources de base de données d'autres applications. Par exemple, une analyse longue de données météorologiques régionales pourrait utiliser temporairement toute la capacité de lecture d'une table . Cette pourrait impacter négativement d'autres applications qui doivent accéder aux mêmes données. Avec DAX, l'analyse météorologique pourrait être effectuée sur des données mises en cache.

DAX n'est pas une solution idéale pour les types d'applications suivants :

- Les applications qui nécessitent des lectures à cohérence forte (ou qui ne peuvent pas tolérer les lectures cohérentes à terme).
- Les applications qui ne nécessitent pas des temps de réponse de l'ordre de la microseconde pour les opérations de lecture, ou qui n'ont pas besoin de décharger une activité de lecture répétée à partir de tables sous-jacentes.
- Applications nécessitant beaucoup d'écriture. Un volume élevé d'écritures entraîne une réplication accrue sur les nœuds DAX d'un cluster. Cela provoque une consommation accrue de ressources et un risque de problèmes de disponibilité.
- Applications sans nombreuses lectures répétées. Les performances du DAX sont optimales lorsque le taux d'accès au cache dépasse 90 %. La baisse des taux d'accès au cache augmente les erreurs de cache, ce qui utilise davantage de ressources dans l'ensemble du cluster DAX.

Notes d'utilisation de DAX

- Pour obtenir la liste des AWS régions dans lesquelles le DAX est disponible, consultez la tarification d'Amazon DynamoDB.
- DAX prend en charge les applications écrites en Go, Java, Node.js, Python et .NET, en utilisant des clients AWS fournis pour ces langages de programmation.

- DAX n'est disponible que pour la plate-forme EC2 -VPC.
- La politique de rôle de service de cluster de DAX doit autoriser l'action `dynamodb:DescribeTable` pour la conservation des métadonnées relatives à la table DynamoDB.
- Les clusters DAX conservent les métadonnées sur les noms d'attribut des éléments qu'ils stockent. Ces métadonnées sont conservées indéfiniment (même après que l'élément a expiré ou été exclu du cache). Les applications qui utilisent un nombre illimité de noms d'attributs peuvent entraîner au fil du temps un épuisement de la mémoire du cluster DAX. Cette limitation s'applique uniquement aux noms d'attributs de niveau supérieur, et non aux noms d'attributs imbriqués. Les noms d'attributs de haut niveau problématiques incluent les horodatages et les sessions UUIDs. IDs

  Cette limitation s'applique uniquement aux noms d'attributs et non à leurs valeurs. Les éléments suivants ne posent pas de problème.

  ```
  {
      "Id": 123,
      "Title": "Bicycle 123",
      "CreationDate": "2017-10-24T01:02:03+00:00"
  }
  ```

  Mais les éléments tels que les suivants en posent s'ils sont suffisamment nombreux et que chacun a un horodatage différent.

  ```
  {
      "Id": 123,
      "Title": "Bicycle 123",
      "2017-10-24T01:02:03+00:00": "created"
  }
  ```

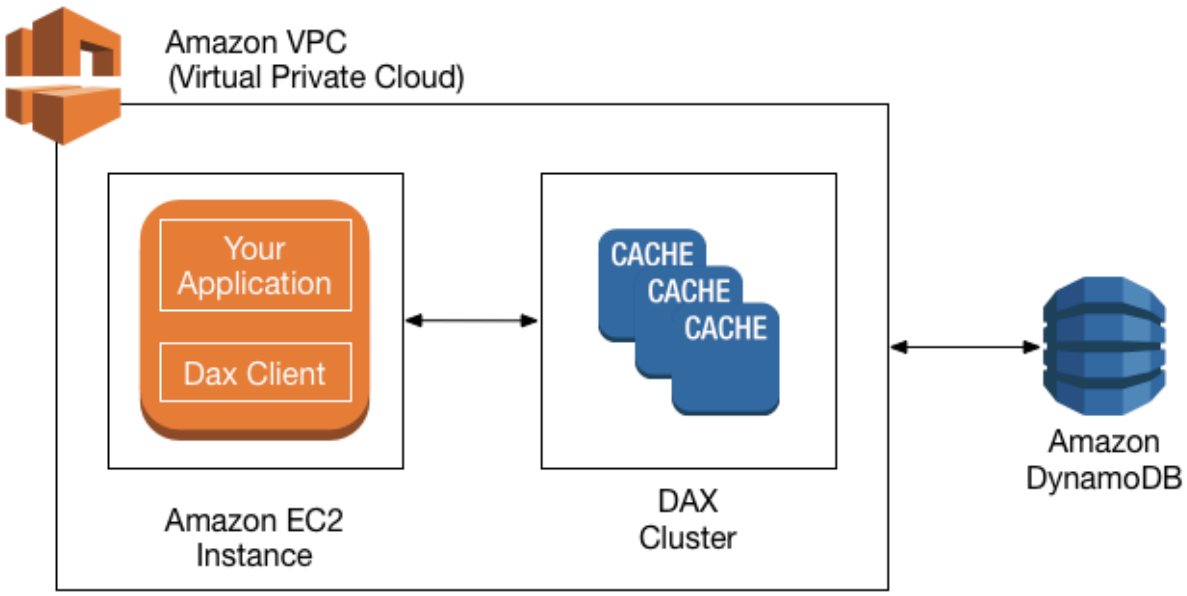
Fonctionnement de DAX

Amazon DynamoDB Accelerator (DAX) est conçu pour s'exécuter dans un environnement Amazon Virtual Private Cloud (Amazon VPC). Le service Amazon VPC définit un réseau virtuel qui ressemble de près à un centre de données classique. Avec un VPC, vous contrôlez la plage d'adresses IP, les sous-réseaux, les tables de routage, les passerelles réseau et les paramètres de sécurité. Vous

pouvez lancer un cluster DAX dans votre réseau privé et contrôler l'accès au cluster à l'aide de groupes de sécurité Amazon VPC.

> **Note**
>
> Si vous avez créé votre AWS compte après le 4 décembre 2013, vous disposez déjà d'un VPC par défaut dans chaque AWS région. Le VPC immédiatement utilisable. Il ne nécessite aucune configuration.
> Pour plus d'informations, consultez VPC par défaut et sous-réseaux par défaut dans le Guide de l'utilisateur Amazon VPC.

Le diagramme suivant est une présentation générale de DAX.



Pour créer un cluster DAX, vous devez utiliser l'AWS Management Console. Sauf spécification contraire, votre cluster DAX s'exécute au sein de votre VPC par défaut. Pour exécuter votre application, vous devez lancer une EC2 instance Amazon dans votre Amazon VPC. Vous déployez ensuite votre application (avec le client DAX) sur l' EC2instance.

Au moment de l'exécution, le client DAX dirige toutes les demandes d'API DynamoDB de votre application vers le cluster DAX. Si DAX peut traiter l'une de ces demandes d'API directement, il le fait. Sinon, il transmet la demande à DynamoDB.

Enfin, le cluster DAX renvoie les résultats à votre application.

Rubriques

- Comment DAX traite les demandes
- Cache d'élément
- Cache de requête

Comment DAX traite les demandes

Un cluster DAX est constitué d'un ou plusieurs nœuds. Chaque nœud exécute sa propre instance du logiciel de mise en cache DAX. L'un des nœuds sert de nœud principal pour le cluster. Les autres nœuds éventuels servent de réplicas en lecture. Pour de plus amples informations, veuillez consulter Nœuds.

Votre application peut accéder à DAX en spécifiant le point de terminaison du cluster DAX. Le logiciel client DAX utilise le point de terminaison du cluster pour effectuer un équilibrage de charge et un routage intelligents.

Opérations de lecture

DAX peut répondre aux appels d'API suivants :

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Si la demande spécifie des lectures éventuellement cohérentes (comportement par défaut), elle tente de lire l'élément à partir de par défaut :

- Si l'élément est disponible dans DAX (accès au cache), DAX le renvoie à l'application sans accéder à DynamoDB.

- Si l'élément n'est pas disponible dans DAX (échec d'accès au cache), DAX transmet la demande à DynamoDB. Après réception de la réponse de DynamoDB, DAX renvoie les résultats à l'application. Mais les résultats sont aussi écrits dans le cache sur le nœud principal.

> **Note**
>
> S'il existe des réplicas en lecture dans le cluster, DAX assure automatiquement leur synchronisation avec le nœud principal. Pour de plus amples informations, veuillez consulter Clusters.

Si la demande spécifie des lectures fortement cohérentes, DAX transmet la demande à DynamoDB. Les résultats de DynamoDB ne sont pas mis en cache dans DAX. Ils sont à la place retournés à l'application.

Opérations d'écriture

Les opérations d'API DAX suivantes sont considérées comme des opérations d'« écriture simultanée » :

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Avec ces opérations, les données sont d'abord écrites dans la table DynamoDB, puis dans le cluster DAX. L'opération ne réussit que si les données sont correctement écrites tant dans la table que dans DAX.

Autres opérations

DAX ne reconnaît pas les opérations DynamoDB pour la gestion des tables (telles que `CreateTable`, `UpdateTable`, etc.). Si votre application a besoin d'effectuer ces opérations, elle doit accéder directement à DynamoDB au lieu d'utiliser DAX.

Pour plus d'informations sur la cohérence entre DAX et DynamoDB, consultez Modèles de cohérence DAX et DynamoDB.

Pour plus d'informations sur le fonctionnement des transactions dans DAX, consultez Utilisation du mode transactionnel APIs dans DynamoDB Accelerator (DAX).

Limitation du débit de demande

Si le nombre de demandes envoyées au DAX dépasse la capacité d'un nœud, le DAX limite le taux d'acceptation de demandes supplémentaires en renvoyant un. ThrottlingException DAX évalue en continu l'utilisation de votre UC pour déterminer le volume de demandes qu'elle peut traiter tout en conservant un état de cluster sain.

Vous pouvez surveiller la ThrottledRequestCount métrique publiée par DAX sur Amazon CloudWatch. Si ces exceptions s'affichent régulièrement, vous devez envisager de mettre à l'échelle votre cluster.

Cache d'élément

DAX gère un cache d'élément pour stocker les résultats des opérations `GetItem` et `BatchGetItem`. Les éléments dans le cache représentent des données éventuellement cohérentes de DynamoDB, et sont stockés sur la base de leurs valeurs de clé primaire.

Quand une application envoie une demande `GetItem` ou `BatchGetItem`, DAX tente de lire les éléments directement dans le cache d'élément à partir des valeurs de clé spécifiées. Si les éléments sont trouvés (accès au cache), DAX les renvoie immédiatement à l'application. Si les éléments ne sont pas trouvés (échec d'accès au cache), DAX envoie la demande à DynamoDB. DynamoDB traite les demandes en utilisant des lectures éventuellement cohérentes et renvoie les éléments à DAX. DAX les stocke dans le cache d'éléments, puis les renvoie à l'application.

Le cache d'éléments dispose d'un paramètre de time-to-live (TTL) dont la valeur par défaut est de 5 minutes. DAX assigne un horodatage à chaque élément qu'il écrit dans le cache d'éléments. Un élément expire s'il reste dans le cache plus longtemps que la durée de vie définie. Une demande `GetItem` émise sur un élément expiré est considérée comme un échec d'accès au cache, et DAX envoie la demande `GetItem` à DynamoDB.

> **Note**
>
> Vous pouvez spécifier le paramètre de durée de vie (TTL) pour le cache d'éléments lorsque vous créez un cluster DAX. Pour de plus amples informations, veuillez consulter Gestion des clusters DAX .

DAX gère par ailleurs une liste LRU (moins récemment utilisé) pour le cache d'éléments. La liste LRU garde une trace du moment où un élément est écrit pour la première fois dans le cache et du moment où il est lu pour la dernière fois dans le cache. Si le cache d'éléments arrive à saturation, DAX expulse les anciens éléments (même s'ils n'ont pas encore expiré) afin de ménager de la place pour les nouveaux. L'algorithme LRU est toujours activé pour le cache d'élément et ne peut pas être configuré par l'utilisateur.

Si vous spécifiez zéro comme paramètre TTL du cache d'éléments, les éléments figurant dans celui-ci ne sont actualisés que suite à une expulsion LRU ou à une opération d'« écriture simultanée ».

Pour plus d'informations sur la cohérence du cache d'éléments dans DAX, consultez Comportement du cache d'éléments DAX.

Cache de requête

DAX gère également un cache de requêtes pour stocker les résultats des opérations `Query` et `Scan`. Les éléments figurant dans ce cache représentent les jeux de résultats des requêtes et des analyses effectuées sur les tables DynamoDB. Ces jeux de résultats sont stockées en fonction des valeurs de leurs paramètres.

Quand une application envoie une demande `Query` ou `Scan`, DAX tente de lire un ensemble de résultats correspondant à partir du cache de requêtes à l'aide des valeurs de paramètres spécifiées. Si le jeu de résultats est trouvé (correspondance dans le cache), le retourne immédiatement à l'application. Si l'ensemble de résultats n'est pas trouvé (échec d'accès au cache), DAX envoie la demande à DynamoDB. DynamoDB traite les demandes en utilisant des lectures éventuellement cohérentes et renvoie l'ensemble de résultats à DAX. DAX stocke celui-ci dans le cache de requêtes, puis le renvoie à l'application.

> **Note**
>
> Vous pouvez spécifier le paramètre de time-to-live (TTL) pour le cache de requêtes lorsque vous créez un cluster DAX. Pour de plus amples informations, veuillez consulter Gestion des clusters DAX .

DAX conserve aussi une liste LRU pour le cache de requêtes. La liste LRU garde une trace du moment où un jeu de résultats est écrit pour la première fois dans le cache et du moment où il est lu pour la dernière fois dans le cache. Si le cache de requêtes arrive à saturation, DAX expulse les

anciens ensembles de résultats (même s'ils n'ont pas encore expiré) afin de ménager de la place pour les nouveaux. L'algorithme LRU est toujours activé pour le cache de requête et ne peut pas être configuré par l'utilisateur.

Si vous spécifiez zéro comme paramètre TTL de cache de requêtes, la réponse à la requête n'est pas mise en cache.

Pour plus d'informations sur la cohérence du cache de requêtes dans DAX, consultez Comportement du cache de requêtes DAX.

Composants de cluster DAX

Un cluster Amazon DynamoDB Accelerator (DAX) est constitué de composants d'infrastructure AWS. Cette section décrit ces composants et explique comment ils fonctionnent ensemble.

Rubriques

- Nœuds
- Clusters
- Régions et Zones de disponibilité
- Groupes de paramètres
- Groupes de sécurité
- ARN de cluster
- Point de terminaison de cluster
- Points de terminaison de nœud
- Groupes de sous-réseaux
- Événements
- Fenêtre de maintenance

Nœuds

Un nœud est le bloc de construction le plus petit d'un cluster . Chaque nœud exécute une instance du logiciel DAX, et gère un réplica unique des données mises en cache.

Vous pouvez mettre à l'échelle votre cluster DAX de deux manières :

- En ajoutant des nœuds supplémentaires au cluster. Cela a pour effet d'accroître le débit de lecture global du cluster.
- En utilisant un type de nœud plus grand. Les types de nœuds de plus grande taille offrent une capacité supérieure et peuvent accroître le débit. (Vous devez créer un nouveau cluster avec le nouveau type de nœud.)

Chaque nœud au sein d'un cluster est de type identique et exécute le même logiciel de mise en cache DAX. Pour accéder à la liste des types de nœuds disponibles, consultez Tarification Amazon DynamoDB.

Clusters

Un cluster est un groupement logique d'un ou plusieurs nœuds que DAX gère en tant qu'unité. L'un des nœuds du cluster est désigné en tant que nœud principal, tandis que les autres nœuds sont des réplicas en lecture (le cas échéant).

Le nœud principal est responsable des éléments suivants :

- Réponse aux demandes d'application pour les données mises en cache.
- Gestion des opérations d'écriture sur DynamoDB.
- Élimination des données du cache, selon la stratégie d'éviction du cluster.

Lorsque des modifications sont apportées aux données mises en cache sur le nœud primaire, DAX les propage à tous les nœuds de réplica en lecture à l'aide des journaux de réplication. Une fois la confirmation reçue de tous les réplicas en lecture, DynamoDB supprime les journaux de réplication du nœud primaire.

Un cluster DAX peut prendre en charge jusqu'à 11 nœuds par cluster (le nœud primaire, plus jusqu'à 10 réplicas en lecture).

Les réplicas en lecture traitent les opérations suivantes :

- Réponse aux demandes d'application pour les données mises en cache.
- Élimination des données du cache, selon la stratégie d'éviction du cluster.

Cependant, contrairement au nœud principal, les réplicas en lecture n'écrivent pas sur DynamoDB.

Les réplicas en lecture remplissent deux fonctions supplémentaires :

- Scalabilité. Si un grand nombre de clients d'application doivent accéder à DAX simultanément, vous pouvez ajouter des réplicas pour la mise à l'échelle de la lecture. DAX répartit uniformément la charge sur tous les nœuds dans le cluster. (Une autre méthode permettant d'accroître le débit consiste à utiliser des types de nœud de cache supérieurs.)
- Haute disponibilité. En cas d'échec du nœud principal, DAX bascule automatiquement vers un réplica en lecture et le désigne en tant que nouveau nœud principal. En cas d'échec d'un nœud réplica, d'autres nœuds dans le cluster DAX peuvent encore servir les demandes jusqu'à la récupération du nœud en échec. Pour bénéficier d'une tolérance aux pannes maximale, vous devez déployer les réplicas en lecture dans des zones de disponibilité distinctes. Cette configuration garantit que votre cluster DAX peut continuer à fonctionner, même si une zone de disponibilité entière devient indisponible.

> **Important**
>
> Pour une utilisation en production, nous vous recommandons vivement d'utiliser DAX avec au moins trois nœuds, chacun d'eux étant placé dans des zones de disponibilité différentes. Trois nœuds sont requis pour qu'un cluster DAX soit tolérant aux défaillances.
> Un cluster DAX peut être déployé avec un ou deux nœuds pour des charges de travail de développement ou de test. Un cluster à un ou deux nœuds n'est pas tolérant aux défaillances ; nous recommandons au moins trois nœuds pour une utilisation en production. Si un cluster à un ou deux nœuds rencontre des erreurs logicielles ou matérielles, il peut devenir indisponible ou perdre des données mises en cache.

> **Important**
>
> Un cluster DAX prend en charge un maximum de 500 tables DynamoDB. Si vous dépassez les 500 tables, la disponibilité et les performances de votre cluster peuvent se dégrader.

Régions et Zones de disponibilité

Un cluster DAX situé dans une région AWS déterminée ne peut interagir qu'avec des tables DynamoDB situées dans la même région. Pour cette raison, vous devez veiller à lancer votre cluster DAX dans la région appropriée. Si vous avez des tables DynamoDB dans d'autres régions, vous devez également lancer des clusters DAX dans celles-ci.

Chaque région est conçue pour être complètement isolée des autres régions . Chaque région dispose de plusieurs zones de disponibilité. En lançant vos nœuds dans différentes zones de disponibilité, vous pouvez obtenir la plus grande tolérance aux pannes possible.

> **Important**
>
> Ne placez pas tous les nœuds de votre cluster dans une seule zone de disponibilité. Dans cette configuration, votre cluster DAX devient indisponible en cas d'échec d'une zone de disponibilité.
> Pour une utilisation en production, nous vous recommandons vivement d'utiliser DAX avec au moins trois nœuds, chacun d'eux étant placé dans des zones de disponibilité différentes. Trois nœuds sont requis pour qu'un cluster DAX soit tolérant aux défaillances.
> Un cluster DAX peut être déployé avec un ou deux nœuds pour des charges de travail de développement ou de test. Un cluster à un ou deux nœuds n'est pas tolérant aux défaillances ; nous recommandons au moins trois nœuds pour une utilisation en production. Si un cluster à un ou deux nœuds rencontre des erreurs logicielles ou matérielles, il peut devenir indisponible ou perdre des données mises en cache.

Groupes de paramètres

Des groupes de paramètres sont utilisés pour gérer les paramètres d'exécution des clusters DAX. DAX propose plusieurs paramètres permettant d'optimiser les performances, par exemple, en définissant une politique TTL pour les données mises en cache. Un groupe de paramètres est une collection nommée de paramètres que vous pouvez appliquer à un cluster. En faisant cela, vous vous assurez que tous les nœuds du cluster sont configurés exactement de la même manière.

Groupes de sécurité

Un cluster DAX s'exécute dans un environnement Amazon Virtual Private Cloud (Amazon VPC). Cet environnement est un réseau virtuel dédié à votre compte AWS et isolé d'autres VPC. Un groupe de sécurité agit en tant que pare-feu virtuel pour votre VPC, vous permettant de contrôler le trafic réseau entrant et sortant.

Lorsque vous lancez un cluster dans votre VPC, vous ajoutez une règle d'entrée à votre groupe de sécurité afin d'autoriser le trafic réseau entrant. La règle d'entrée spécifie le protocole (TCP) et le numéro de port (8111) pour votre cluster. Une fois cette règle ajoutée à votre groupe de sécurité, les applications s'exécutant dans votre VPC peuvent accéder au cluster DAX.

ARN de cluster

Un Amazon Resource Name (ARN) est attribué à chaque cluster DAX. Le format de l'ARN est le suivant.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Vous utilisez l'ARN du cluster dans une politique IAM pour définir les autorisations pour des opérations d'API DAX. Pour plus d'informations, consultez Contrôle d'accès à DAX.

Point de terminaison de cluster

Chaque cluster DAX fournit un point de terminaison de cluster que votre application peut utiliser. En accédant au cluster à l'aide de ce point de terminaison, votre application n'a pas besoin de connaître les noms d'hôte et les numéros de port des nœuds individuels du cluster. Votre application « connaît » automatiquement tous les nœuds du cluster, même si vous ajoutez ou supprimez des réplicas en lecture.

Voici un exemple de point de terminaison de cluster dans la région us-east-1 qui n'est pas configuré pour utiliser le chiffrement en transit.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Voici un exemple de point de terminaison de cluster dans la même région qui est configuré pour utiliser le chiffrement en transit.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Points de terminaison de nœud

Chacun des nœuds dans un cluster DAX a ses propres nom d'hôte et numéro de port. Voici un exemple de point de terminaison d'un nœud.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

Votre application peu accéder directement à un nœud à l'aide de son point de terminaison. Cependant, nous vous recommandons de traiter le cluster DAX en tant qu'unité unique, et d'y accéder en utilisant le point de terminaison de cluster à la place. Le point de terminaison de cluster permet à votre application de ne pas maintenir une liste de nœuds et de garder cette liste à jour lorsque vous ajoutez ou supprimez des nœuds dans le cluster.

Groupes de sous-réseaux

L'accès aux nœuds d'un cluster DAX est limité aux applications s'exécutant sur des instances Amazon EC2 au sein d'un environnement de VPC Amazon. Vous pouvez utiliser des groupes de sous-réseaux pour accorder l'accès au cluster à partir d'instances Amazon EC2 s'exécutant sur des sous-réseaux spécifiques. Un groupe de sous-réseaux est un ensemble de sous-réseaux (généralement privés) que vous pouvez utiliser pour vos clusters fonctionnant dans un environnement de VPC Amazon.

Lorsque vous créez un cluster DAX, vous devez spécifier un groupe de sous-réseaux. DAX utilise ce groupe de sous-réseaux de cache pour sélectionner un sous-réseau et des adresses IP au sein de celui-ci à associer à vos nœuds.

Événements

DAX enregistre les événements importants qui se produisent au sein de vos clusters, tels que l'échec ou la réussite de l'ajout d'un nœud, ou des modifications apportées à des groupes de sécurité. En surveillant ces événements clés, vous pouvez connaître l'état actuel de vos clusters et, selon l'événement, vous pouvez prendre une mesure corrective. Vous pouvez accéder à ces événements à l'aide de l'action AWS Management Console ou `DescribeEvents` dans l'API de gestion DAX.

Vous pouvez également demander l'envoi de ces notifications à une rubrique Amazon Simple Notification Service (Amazon SNS) spécifique. Vous serez alors immédiatement informé de tout événement se produisant dans votre cluster DAX.

Fenêtre de maintenance

Chaque cluster a un créneau de maintenance hebdomadaire pour l'application des modifications systèmes. Au fur et à mesure que les modifications sont appliquées de manière séquentielle, un nœud existant est remplacé et un nouveau nœud contenant les modifications appliquées est ajouté au cluster. Au cours de cette période, votre application peut être confrontée à des erreurs ou à des ralentissements transitoires. Par conséquent, nous vous recommandons de planifier la période de maintenance pendant la période d'utilisation la plus faible et de modifier ce calendrier périodiquement selon les besoins. Vous pouvez spécifier une plage de temps de 24 heures au cours de laquelle toutes les opérations de maintenance demandées doivent avoir lieu.

Si vous ne spécifiez pas de fenêtre de maintenance souhaitée lors de la création ou de la modification d'un cluster de cache, DAX affecte une fenêtre de maintenance de 60 minutes sur un

jour de la semaine pris au hasard. Cette fenêtre de maintenance de 60 minutes est sélectionnée de manière aléatoire sur un bloc de temps de 8 heures pour chaque Région AWS. Le tableau suivant répertorie pour les différentes régions les blocs de temps à partir desquels les créneaux de maintenance par défaut sont alloués.

Code région	Nom de la région	Fenêtre de maintenance
ap-northeast-1	Région Asie-Pacifique (Tokyo)	13:00–21:00 UTC
ap-southeast-1	Région Asie-Pacifique (Singapour)	14:00–22:00 UTC
ap-southeast-2	Région Asie-Pacifique (Sydney)	12:00–20:00 UTC
ap-south-1	Région Asie-Pacifique (Mumbai)	17:30–1:30 UTC
cn-northwest-1	Région Chine (Ningxia)	23:00–07:00 UTC
cn-north-1	Région Chine (Beijing)	14:00–22:00 UTC
eu-central-1	Région Europe (Francfort)	23:00–07:00 UTC
eu-north-1	Région Europe (Stockholm)	01:00–09:00 UTC
eu-south-2	Région Europe (Espagne)	21:00–05:00 UTC
eu-west-1	Région Europe (Irlande)	22:00–06:00 UTC
eu-west-2	Région Europe (Londres)	23:00–07:00 UTC
eu-west-3	Région Europe (Paris)	23:00–07:00 UTC
sa-east-1	Région Amérique du Sud (São Paulo)	01:00–09:00 UTC
us-east-1	Région USA Est (Virginie du Nord)	03:00–11:00 UTC
us-east-2	Région USA Est (Ohio)	23:00–07:00 UTC

Code région	Nom de la région	Fenêtre de maintenance
us-west-1	Région US West (N. Californi a)	06:00–14:00 UTC
us-west-2	Région USA Ouest (Oregon)	06:00–14:00 UTC

Création d'un cluster DAX

Cette section vous guide tout au long des étapes de configuration et d'utilisation initiales d'Amazon DynamoDB Accelerator (DAX) dans votre environnement Amazon Virtual Private Cloud (Amazon VPC) par défaut. Vous pouvez créer votre premier cluster DAX en utilisant le AWS Command Line Interface(AWS CLI) ou le AWS Management Console.

Après avoir créé votre cluster DAX, vous pouvez y accéder depuis une EC2 instance Amazon exécutée dans le même VPC. Vous pouvez ensuite utiliser votre cluster DAX avec un programme d'application. Pour plus d'informations, consultez Développement avec le client DynamoDB Accelerator (DAX).

Rubriques

- Création d'une fonction du service IAM pour permettre à DAX d'accéder à DynamoDB
- DAX et IPv6
- Création d'un cluster DAX à l'aide du AWS CLI
- Création d'un cluster DAX à l'aide de la AWS Management Console

Création d'une fonction du service IAM pour permettre à DAX d'accéder à DynamoDB

Pour permettre à votre cluster DAX d'accéder à vos tables DynamoDB en votre nom, vous devez créer un rôle de service. Un rôle de service est un rôle Gestion des identités et des accès AWS(IAM) qui autorise un AWS service à agir en votre nom. Le rôle de service permet à DAX d'accéder à vos tables DynamoDB, comme si vous-même y accédiez vous-même. Vous devez créer le rôle de service avant de créer le cluster DAX.

Si vous utilisez la console, le flux de travail dédié à la création d'un cluster vérifie la présence d'un rôle de service DAX pré-existant. Si aucun rôle de service n'est trouvé, la console crée un nouveau

rôle de service pour vous. Pour de plus amples informations, veuillez consulter the section called “Étape 2 : créer un cluster DAX”.

Si vous utilisez le AWS CLI, vous devez spécifier un rôle de service DAX que vous avez créé précédemment. Sinon, vous devez créer un nouveau rôle de service au préalable. Pour de plus amples informations, veuillez consulter Étape 1 : créer un rôle de service IAM pour que DAX accède à DynamoDB à l'aide du AWS CLI.

Autorisations requises pour créer une fonction du service

La politique `AdministratorAccess` gérée par AWS fournit toutes les autorisations nécessaires à la création d’un cluster DAX et d’un rôle de service. Si votre utilisateur a joint `AdministratorAccess`, aucune autre action n’est nécessaire.

Autrement, vous devez ajouter les autorisations suivantes à votre politique IAM afin que votre utilisateur soit en mesure de créer la fonction du service :

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Vous devez attacher ces autorisations à l’utilisateur qui tente de réaliser cette action.

> **Note**
>
> Les `iam:PassRole` autorisations `iam:CreateRoleiam:CreatePolicy`,`iam:AttachRolePolicy`, et ne sont pas incluses dans les politiques AWS gérées pour DynamoDB. La conception en est la raison : ces autorisations fournissent la possibilité d’une escalade de privilège. En effet, un utilisateur peut utiliser ces autorisations pour créer une nouvelle politique d’administrateur puis l’attacher à un rôle existant. Ainsi, vous (administrateur de votre cluster DAX) devez ajouter explicitement ces autorisations à votre politique.

Résolution des problèmes

Si votre politique utilisateur ne dispose pas des autorisations `iam:CreateRole`, `iam:CreatePolicy` et `iam:AttachPolicy`, vous rencontrerez des messages d'erreur. Le tableau ci-dessous illustre ces messages et la manière de corriger les problèmes.

Si vous obtenez ce message d'erreur...	Procédez comme suit :
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: accountID :role/service-role/ roleName`	Ajoutez `iam:CreateRole` à votre politique utilisateur.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreatePolicy on resource: policy policyName`	Ajoutez `iam:CreatePolicy` à votre politique utilisateur.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:AttachRolePolicy on resource: role daxServiceRole`	Ajoutez `iam:AttachRolePolicy` à votre politique utilisateur.

Pour plus d'informations sur les politiques IAM requises pour l'administration de cluster DAX, consultez Contrôle d'accès à DAX.

DAX et IPv6

DynamoDB DAX IPv6 prend désormais en charge l'adressage, ce qui vous permet de créer des clusters qui fonctionnent IPv4 en mode réseau uniquement IPv6, uniquement ou à double pile. Cela permet d'améliorer les capacités de mise en réseau pour répondre aux exigences évolutives de l'infrastructure.

Types de réseaux :

Vous pouvez créer des clusters avec les types de réseaux suivants :

- IPv4-uniquement

- IPv6-uniquement
- Double pile (prend en charge les deux IPv4 et IPv6)

Caractéristiques principales :

Avec IPv6 l'assistance, vous pouvez effectuer les opérations suivantes :

- Options de configuration réseau :
  - IPv4-uniquement et clusters à double pile activés. `dual_stack subnets`
  - IPv6clusters -only sur des sous-réseaux IPv6 -only.
- Gestion des groupes de sous-réseaux :
  - Créez des groupes de sous-réseaux avec IPv4 prise en charge uniquement, IPv6 uniquement ou double pile
  - Modifier les groupes de sous-réseaux existants avec des sous-réseaux VPC supplémentaires
  - Ajouter des sous-réseaux IPv6 uniquement aux groupes de sous-réseaux IPv6 configurés
  - Ajouter des sous-réseaux IPv4 ou des sous-réseaux à double pile IPv4 et des groupes configurés à double pile
- Configuration du client :
  - Lorsque vous effectuez des appels de plan de données, vous pouvez définir le protocole IP préféré pour les clusters dual_stack en utilisant :
    - `ip_discovery`paramètre dans le SDK Python
    - `ipDiscovery`paramètre dans un autre SDKs
  - Par défaut : IPv4 lorsque la préférence de protocole n'est pas spécifiée

Avant de procéder IPv6 à l'implémentation dans vos clusters DAX, vous devez prendre en compte les points suivants :

- Le type de réseau ne peut pas être modifié après la création du cluster
- Pour les clusters à double pile, le `ip_discovery/ipDiscovery` paramètre de la configuration du client détermine le protocole IP à utiliser (IPv4 ou IPv6)
- Différentes applications peuvent se connecter au même cluster à double pile en utilisant différents protocoles IP en fonction de leur configuration

Example Exemple de configuration client

```
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(endpoint)             // DAX cluster endpoint
            .ipDiscovery(ipDiscovery)       // IP discovery type (IPv4 or IPv6)
            .build())
        .build();
```

Important

Lorsque vous utilisez des politiques IAM basées sur les ressources pour restreindre les adresses IP des tables DynamoDB dans des environnements DAX IPv6 uniquement, vous devez créer une exception pour le rôle IAM de votre cluster DAX si vous bloquez l'espace d'adressage (). IPv4 `0.0.0.0/0` Ajoutez une `ArnNotEquals` condition à votre politique qui autorise spécifiquement l'accès au rôle IAM du cluster DAX tout en maintenant les restrictions basées sur l'IP pour les autres chemins d'accès. Sans cette exception, DAX ne peut pas accéder à votre table DynamoDB.
Par exemple :

Example

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
DAXServiceRoleForDynamoDBAccess"
        },
        "IpAddress": {
          "aws:SourceIp": "0.0.0.0/0"
```

```
        }
      }
    }
  ]
}
```

Création d'un cluster DAX à l'aide du AWS CLI

Cette section explique comment créer un cluster Amazon DynamoDB Accelerator (DAX) à l'aide de l'AWS Command Line Interface(AWS CLI). Si vous ne l'avez pas déjà fait, vous devez installer et configurer AWS CLI. Pour ce faire, consultez les instructions suivantes dans le Guide de l'utilisateur AWS Command Line Interface :

- Installation du AWS CLI
- Configuration de l'interface AWS CLI (français non garanti)

> **Important**
>
> Pour gérer les clusters DAX à l'aide de AWS CLI, installez ou effectuez une mise à niveau vers la version 1.11.110 ou supérieure.

Tous les AWS CLI exemples utilisent la `us-west-2` région et un compte fictif. IDs

Rubriques

- Étape 1 : créer un rôle de service IAM pour que DAX accède à DynamoDB à l'aide du AWS CLI
- Étape 2 : créer un groupe de sous-réseaux
- Étape 3 : créer un cluster DAX à l'aide du AWS CLI
- Étape 4 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS CLI

Étape 1 : créer un rôle de service IAM pour que DAX accède à DynamoDB à l'aide du AWS CLI

Avant de créer un cluster Amazon DynamoDB Accelerator (DAX), vous devez créer un rôle de service pour celui-ci. Un rôle de service est un rôle Gestion des identités et des accès AWS(IAM)

qui autorise un AWS service à agir en votre nom. Le rôle de service permet à DAX d'accéder à vos tables DynamoDB comme si vous y accédiez vous-même.

Au cours de cette étape, vous créez une politique IAM et l'attachez à un rôle IAM. Vous pourrez ainsi attribuer le rôle à un cluster DAX pour permettre à celui-ci d'effectuer des opérations DynamoDB en votre nom.

Pour créer un rôle de service IAM pour DAX

1. Créez un fichier nommé `service-trust-relationship.json` avec les contenus suivants.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
          {
               "Effect": "Allow",
               "Principal": {
                   "Service": "dax.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. Crée le rôle de service.

   ```
   aws iam create-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --assume-role-policy-document file://service-trust-relationship.json
   ```

3. Créez un fichier nommé `service-role-policy.json` avec les contenus suivants.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Action": [
   ```

```
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

Remplacez *accountID* par votre identifiant de AWS compte. Pour trouver votre identifiant de AWS compte, dans le coin supérieur droit de la console, choisissez votre identifiant de connexion. L'identifiant de votre AWS compte apparaît dans le menu déroulant.

Dans l'exemple, le nom de ressource Amazon (ARN) *accountID* doit être un nombre à 12 chiffres. N'utilisez aucun trait d'union ou autre ponctuation.

4. Créez une politique IAM pour le rôle de service.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

Dans la sortie, notez l'ARN de la politique que vous avez créée, comme dans l'exemple suivant.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Attache la politique au rôle de service. Remplacez *arn* le code suivant par l'ARN du rôle réel de l'étape précédente.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
```

```
    --policy-arn arn
```

Ensuite, spécifiez un groupe de sous-réseau pour votre VPC par défaut. Un groupe de sous-réseaux est un ensemble constitué d'un ou plusieurs sous-réseaux au sein de votre VPC. Consultez Étape 2 : créer un groupe de sous-réseaux.

Étape 2 : créer un groupe de sous-réseaux

Suivez cette procédure pour créer un groupe de sous-réseaux pour votre cluster Amazon DynamoDB Accelerator (DAX) à l'aide du ().AWS Command Line InterfaceAWS CLI

> **Note**
>
> Si vous avez déjà créé un groupe de sous-réseaux pour votre VPC par défaut, vous pouvez passer cette étape.

DAX est conçu pour s'exécuter dans un environnement Amazon Virtual Private Cloud (Amazon VPC). Si vous avez créé votre compte AWS après le 4 décembre 2013, vous disposez déjà d'un VPC par défaut dans chaque région AWS. Pour plus d'informations, consultez VPC par défaut et sous-réseaux par défaut dans le Guide de l'utilisateur Amazon VPC.

Pour créer un groupe de sous-réseaux

1. Pour déterminer l'identificateur de votre VPC par défaut, tapez la commande suivante.

   ```
   aws ec2 describe-vpcs
   ```

   Dans la sortie, notez l'identificateur de votre VPC par défaut, comme dans l'exemple suivant.

   ```
   vpc-12345678
   ```

2. Déterminez le sous-réseau IDs associé à votre VPC par défaut. *vpcID*Remplacez-le par votre identifiant VPC actuel, par exemple,. `vpc-12345678`

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

Dans la sortie, notez les identifiants de sous-réseau, par exemple, `subnet-11111111`.

3. Créez un groupe de sous-réseau. Veillez à spécifier au moins un identifiant de sous-réseau dans le paramètre `--subnet-ids`.

```
aws dax create-subnet-group \
    --subnet-group-name my-subnet-group \
    --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Pour créer le cluster, consultez Étape 3 : créer un cluster DAX à l'aide du AWS CLI.

Étape 3 : créer un cluster DAX à l'aide du AWS CLI

Suivez cette procédure pour utiliser le AWS Command Line Interface(AWS CLI) afin de créer un cluster Amazon DynamoDB Accelerator (DAX) dans votre Amazon VPC par défaut.

Pour créer un cluster DAX

1. Obtenez l'Amazon Resource Name (ARN) pour votre rôle de service.

```
aws iam get-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --query "Role.Arn" --output text
```

Dans la sortie, notez l'ARN de rôle de service, comme dans l'exemple suivant.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. Créez le cluster DAX. Remplacez *roleARN* par l'ARN de l'étape précédente.

```
aws dax create-cluster \
    --cluster-name mydaxcluster \
    --node-type dax.r4.large \
    --replication-factor 3 \
    --iam-role-arn roleARN \
    --subnet-group my-subnet-group \
    --sse-specification Enabled=true \
    --region us-west-2
```

Tous les nœuds du cluster sont de type `dax.r4.large` (`--node-type`). Il y a trois nœuds (`--replication-factor`), un nœud primaire et deux réplicas.

> **Note**
>
> `sudo` et `grep` étant des mots clés réservés, vous ne pouvez pas créer un cluster DAX avec ces mots dans le nom du cluster. Par exemple, `sudo` et `sudocluster` sont des noms de cluster non valides.

Pour afficher le statut du cluster, tapez la commande suivante.

```
aws dax describe-clusters
```

Le statut est indiqué dans la sortie, par exemple, `"Status": "creating"`.

> **Note**
>
> La création du cluster prendra plusieurs minutes. Lorsque le cluster est prêt, son statut passe à `available`. Pendant ce temps, vous pouvez passer à Étape 4 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS CLI et suivre les instructions qui y sont fournies.

Étape 4 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS CLI

Les nœuds dans votre cluster Amazon DynamoDB Accelerator (DAX) utilisent le groupe de sécurité par défaut pour votre VPC Amazon. Pour le groupe de sécurité par défaut, vous devez autoriser le trafic entrant sur le port TCP 8111 pour les clusters non chiffrés, ou le port 9111 pour les clusters chiffrés. Cela permet aux EC2 instances Amazon de votre Amazon VPC d'accéder à votre cluster DAX.

> **Note**
>
> Si vous avez lancé votre cluster DAX avec un groupe de sécurité différent (autre que `default`), vous devez plutôt suivre cette procédure pour ce groupe.

Pour configurer les règles de trafic entrant du groupe de sécurité

1. Pour déterminer l'identificateur du groupe de sécurité par défaut, tapez la commande suivante. Remplacez *vpcID* par votre identifiant VPC réel (depuis Étape 2 : créer un groupe de sous-réseaux).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   Dans la sortie, prenez note de l'identifiant du groupe de sécurité, par exemple, `sg-01234567`.

2. Entrez ce qui suit. Remplacez la valeur *sgID* par l'identificateur de votre groupe de sécurité réel. Utilisez le port 8111 pour les clusters non chiffrés, et le port 9111 pour les clusters chiffrés.

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

Création d'un cluster DAX à l'aide de la AWS Management Console

Cette section explique comment créer un cluster Amazon DynamoDB Accelerator (DAX) à l'aide de l'AWS Management Console.

Rubriques

- Étape 1 : créer un groupe de sous-réseaux à l'aide du AWS Management Console
- Étape 2 : créer un cluster DAX à l'aide du AWS Management Console
- Étape 3 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS Management Console

Étape 1 : créer un groupe de sous-réseaux à l'aide du AWS Management Console

Suivez cette procédure afin de créer un groupe de sous-réseaux pour votre cluster Amazon DynamoDB Accelerator (DAX) à l'aide de l'AWS Management Console.

> **Note**
>
> Si vous avez déjà créé un groupe de sous-réseaux pour votre VPC par défaut, vous pouvez passer cette étape.

DAX est conçu pour s'exécuter dans un environnement Amazon Virtual Private Cloud (Amazon VPC). Si vous avez créé votre compte AWS après le 4 décembre 2013, vous disposez déjà d'un VPC par défaut dans chaque région AWS. Pour plus d'informations, consultez VPC par défaut et sous-réseaux par défaut dans le Guide de l'utilisateur Amazon VPC.

Dans le cadre du processus de création d'un cluster DAX, vous devez spécifier un groupe de sous-réseaux. Un groupe de sous-réseaux est un ensemble constitué d'un ou plusieurs sous-réseaux au sein de votre VPC. Lorsque vous créez votre cluster DAX, les nœuds sont déployés dans les sous-réseaux du groupe de sous-réseaux.

> **Note**
>
> Le VPC doté de ce cluster DAX peut contenir d'autres ressources et même des points de terminaison VPC pour les autres services, à l'exception du point de terminaison VPC pour les opérations du cluster DAX, ce qui peut entraîner des erreurs dans ElastiCache les opérations du cluster DAX.

Pour créer un groupe de sous-réseaux

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation, sous DAX, choisissez Groupes de sous-réseaux.
3. Choisissez Créer groupe de sous-réseaux.
4. Dans la fenêtre Créer groupe de sous-réseaux, procédez comme suit :
    a. Nom – Saisissez un nom pour le groupe de sous-réseaux.
    b. Description – Saisissez une description pour le groupe de sous-réseaux.
    c. VPC ID (ID de VPC) – Choisissez l'identifiant de votre environnement Amazon VPC.
    d. Subnets (Sous-réseaux) – Sélectionnez un ou plusieurs sous-réseaux dans la liste.

> **Note**
>
> Les sous-réseaux sont répartis sur plusieurs zones de disponibilité. Si vous envisagez de créer un cluster DAX à plusieurs nœuds (un nœud principal et une ou plusieurs répliques de lecture), nous vous recommandons de choisir plusieurs sous-réseaux. IDs DAX peut alors déployer les nœuds de cluster dans plusieurs zones de disponibilité. Si une zone de disponibilité devient indisponible, DAX bascule automatiquement vers une zone de disponibilité survivante. Votre cluster DAX continue de fonctionner sans interruption.

Lorsque les paramètres vous conviennent, cliquez sur Create subnet group (Créer un groupe de sous-réseaux).

Pour créer le cluster, consultez Étape 2 : créer un cluster DAX à l'aide du AWS Management Console.

Étape 2 : créer un cluster DAX à l'aide du AWS Management Console

Suivez cette procédure afin de créer un cluster Amazon DynamoDB Accelerator (DAX) dans votre VPC Amazon par défaut.

Pour créer un cluster DAX

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation, sous DAX, choisissez Clusters.
3. Choisissez Créer un cluster.
4. Dans la fenêtre Créer un cluster, procédez comme suit :

   a. Cluster name (Nom du cluster) – Saisissez un nom court pour votre cluster DAX.

> **Note**
>
> `sudo` et `grep` étant des mots clés réservés, vous ne pouvez pas créer un cluster DAX avec ces mots dans le nom du cluster. Par exemple, `sudo` et `sudocluster` sont des noms de cluster non valides.

b. Cluster description (Description du cluster) – Saisissez une description pour le cluster.

c. Type de nœud—Choisissez le type de nœud pour tous les nœuds du cluster.

d. Cluster size (Taille du cluster) – Choisissez le nombre de nœuds du cluster. Un cluster se compose d'un nœud principal et d'un maximum de neuf réplicas en lecture.

> **Note**
>
> Si vous souhaitez créer un cluster à un seul nœud, choisissez 1. Votre cluster sera composé d'un nœud principal.
> Si vous souhaitez créer un cluster à plusieurs nœuds, choisissez un nombre compris entre 3 (un principal et deux réplicas en lecture) et 10 (un principal et neuf réplicas en lecture).
>
> > **Important**
> >
> > Pour une utilisation en production, nous recommandons vivement d'utiliser DAX avec au moins trois nœuds, chacun étant placé dans une zone de disponibilité distincte. Trois nœuds sont requis pour qu'un cluster DAX soit tolérant aux défaillances.
> > Un cluster DAX peut être déployé avec un ou deux nœuds pour des charges de travail de développement ou de test. Un cluster à un ou deux nœuds n'est pas tolérant aux défaillances ; nous recommandons au moins trois nœuds pour une utilisation en production. Si un cluster à un ou deux nœuds rencontre des erreurs logicielles ou matérielles, il peut devenir indisponible ou perdre des données mises en cache.

e. Choisissez Suivant.

f. Groupe de sous-réseaux—Sélectionnez Choisir un modèle existant et choisissez le groupe de sous-réseaux que vous avez créé dans Étape 1 : créer un groupe de sous-réseaux à l'aide du AWS Management Console.

g. Contrôle d'accès— Choisissez le groupe de sécurité par défaut.

h. Zones de disponibilité (AZ)—ChoisissezAutomatique.

i. Choisissez Suivant.

j. IAM service role for DynamoDB access (Rôle de service IAM pour l'accès à DynamoDB) – Choisissez Create new (Créer), puis saisissez les informations suivantes :

- IAM role name (Nom de rôle IAM) – Saisissez un nom pour un rôle IAM, par exemple, `DAXServiceRole`. La console crée un rôle IAM et votre cluster DAX assume ce rôle lors de l'exécution.
- Cochez la case à côté de Créer une politique.
- IAM role policy (Politique de rôle IAM) – Choisissez Read/Write (Lecture/écriture). Cela permet au cluster DAX d'effectuer des opérations de lecture et d'écriture dans DynamoDB.
- Nouveau nom de politique IAM – Ce champ sera renseigné au fur et à mesure que vous saisissez le nom du rôle IAM. Vous pouvez également saisir le nom d'une politique IAM, par exemple, `DAXServicePolicy`. La console crée une politique IAM et l'attache au rôle IAM.
- Accès aux tables DynamoDB – Choisissez Toutes les tables.

k. Chiffrement – Choisissez Activer le chiffrement au repos et Activer le chiffrement en transit. Pour plus d'informations, consultez Chiffrement au repos DAX et Chiffrement DAX en transit.

Un rôle de service distinct EC2 est également requis pour permettre à DAX d'accéder à Amazon. DAX crée automatiquement ce rôle de service pour vous. Pour en savoir plus, consultez Utilisation des rôles liés à un service pour DAX.

5. Lorsque les paramètres vous conviennent, choisissez Suivant.
6. Groupe de paramètres – Choisissez Choisir un groupe existant.
7. Fenêtre de maintenance – Choisissez Aucune préférence si la fenêtre d'application des mises à jour logicielles vous importe peu, ou choisissez Spécifier une fenêtre horaire et renseignez les options Jour de la semaine, Heure (UTC) et Commencer dans (heures) pour planifier votre fenêtre de maintenance.
8. Balises —Choisissez Ajouter une nouvelle balise pour saisir une key/value paire à des fins de balisage.
9. Choisissez Suivant.

Vous pouvez consulter tous les paramètres sur l'écran Vérifier et créer. Si vous êtes prêt à créer le cluster, choisissez Créer un cluster.

Sur l'écran Clusters, votre cluster DAX est associé à l'état Creating (Création en cours).

> **Note**
>
> La création du cluster prendra plusieurs minutes. Lorsque le cluster est prêt, son statut passe à Available (Disponible).
> Pendant ce temps, vous pouvez passer à Étape 3 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS Management Console et suivre les instructions qui y sont fournies.

Étape 3 : configurer les règles entrantes du groupe de sécurité à l'aide du AWS Management Console

Votre cluster Amazon DynamoDB Accelerator (DAX) communique via le port TCP 8111 (pour les clusters non chiffrés) ou le port 9111 (pour les clusters chiffrés). Vous devez donc autoriser le trafic entrant sur ce port. Cela permet aux EC2 instances Amazon de votre Amazon VPC d'accéder à votre cluster DAX.

> **Note**
>
> Si vous avez lancé votre cluster DAX avec un groupe de sécurité différent (autre que `default`), vous devez plutôt suivre cette procédure pour ce groupe.

Pour configurer les règles de trafic entrant du groupe de sécurité

1. Ouvrez la EC2 console Amazon à l'adresse https://console.aws.amazon.com/ec2/.
2. Dans le panneau de navigation, choisissez Groupes de sécurité.
3. Choisissez le groupe de sécurité par défaut. Dans le menu Actions, choisissez Edit inbound rules (Modifier les règles entrantes).
4. Choisissez Ajouter une règle et entrez les informations suivantes :
   - Port Range (Plage de ports) – Saisissez 8111 (si votre cluster n’est pas chiffré) ou 9111 (si votre cluster est chiffré).
   - Source – Conservez la valeur Personnalisé, puis cliquez sur le champ de recherche à droite. Un menu déroulant s’affiche. Choisissez l’identifiant de votre groupe de sécurité par défaut.
5. Choisissez Enregistrer les règles pour enregistrer les changements.

6. Pour mettre à jour le nom dans la console, accédez à la propriété Nom et choisissez l'option Modifier qui s'affiche.

Modèles de cohérence DAX et DynamoDB

Amazon DynamoDB Accelerator (Dax) est un service de mise en cache à écriture simultanée conçu pour simplifier l'ajout d'un cache à des tables DynamoDB. DAX opérant séparément de DynamoDB, il est important que vous compreniez les modèles de cohérence de DAX et de DynamoDB pour vous assurer que vos applications se comportent comme prévu.

Dans de nombreux cas d'utilisation, la façon dont une application utilise DAX a une incidence sur la cohérence des données au sein du cluster DAX, ainsi qu'entre DAX et DynamoDB.

Rubriques

- Cohérence entre les nœuds d'un cluster DAX
- Comportement du cache d'éléments DAX
- Comportement du cache de requêtes DAX
- Lectures transactionnelles à cohérence forte
- Mise en cache négative
- Politiques pour les écritures

Cohérence entre les nœuds d'un cluster DAX

Pour que votre application bénéficie d'une haute disponibilité, nous vous recommandons d'allouer au moins trois nœuds à votre cluster DAX. Puis, placez ces nœuds dans plusieurs zones de disponibilité d'une région.

Lorsqu'il s'exécute, votre cluster DAX réplique ainsi les données entre tous les nœuds du cluster (sous réserve que vous en ayez provisionné plusieurs). Prenons l'exemple d'une application qui accomplit une opération `UpdateItem` à l'aide de DAX. Cette action entraîne la modification du cache d'éléments du nœud principal avec la nouvelle valeur. Cette valeur est ensuite répliquée sur tous les autres nœuds du cluster. Cette réplication est cohérente à terme et son exécution prend généralement moins d'une seconde.

Dans ce scénario, il est possible que deux clients lisent la même clé sur le même cluster DAX, mais reçoivent des valeurs différentes selon le nœud auquel chaque client accède. Les nœuds seront tous

cohérents une fois que la mise à jour aura été entièrement répliquée sur tous les nœuds du cluster. (Ce comportement est similaire à celui de la cohérence éventuelle de DynamoDB.)

Si vous créez une application qui utilise DAX, cette application doit être conçue de façon à pouvoir tolérer des données éventuellement cohérentes.

Comportement du cache d'éléments DAX

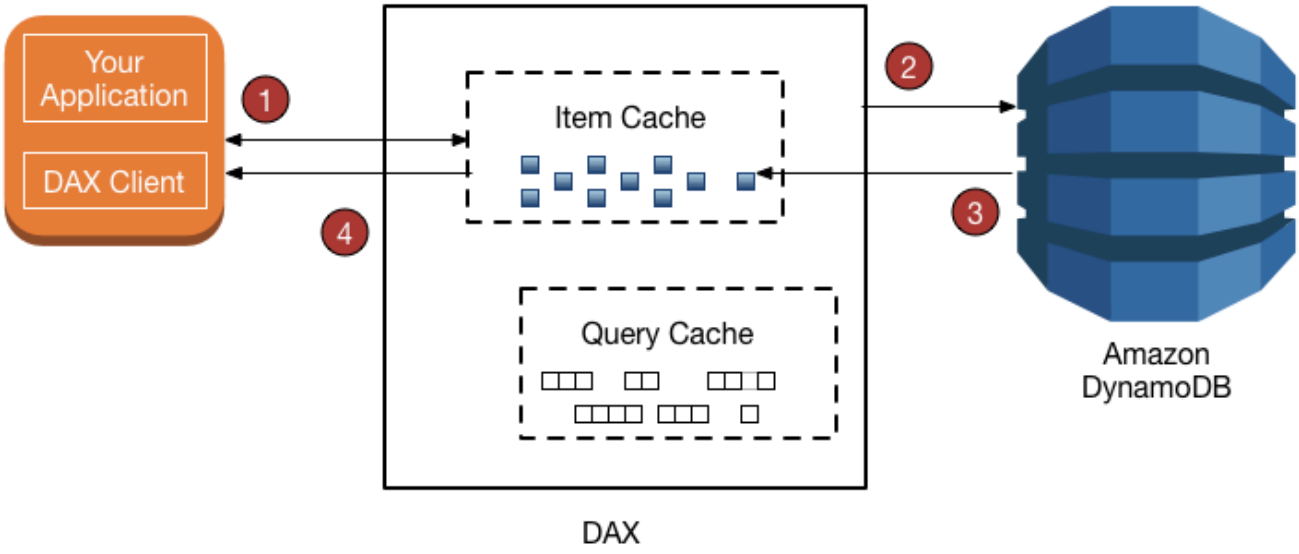
Chaque cluster DAX est doté de deux caches : un cache d'éléments et un cache de requêtes. Pour de plus amples informations, veuillez consulter Fonctionnement de DAX.

Cette section a trait aux implications en termes de cohérence de la lecture et de l'écriture dans le cache d'éléments DAX.

Cohérence des lectures

Avec DynamoDB, par défaut, l'opération `GetItem` effectue une lecture éventuellement cohérente. Supposons que vous utilisez `UpdateItem` avec le client DynamoDB. Si vous tentez de lire le même élément tout de suite après, les données se présenteront peut-être comme avant la mise à jour. Cela est dû au temps nécessaire à la propagation vers tous les emplacements de stockage DynamoDB. La cohérence est généralement atteinte en quelques secondes. Si vous tentez à nouveau de lire, il est probable que vous verrez l'élément mis à jour.

Lorsque vous utilisez `GetItem` avec le client DAX, l'opération (dans ce cas, une lecture éventuellement cohérente) se déroule comme suit.



1. Le client DAX émet une demande `GetItem`. DAX essaie de lire l'élément demandé dans le cache d'éléments. Si l'élément se trouve dans le cache (accès au cache), DAX le renvoie à l'application.
2. Si l'élément n'est pas disponible (échec d'accès au cache), DAX effectue une opération `GetItem` éventuellement cohérente par rapport à DynamoDB.
3. DynamoDB renvoie l'élément demandé que DAX stocke dans le cache d'éléments.
4. DAX renvoie l'élément à l'application.
5. (Non illustré) Si le cluster DAX contient plusieurs nœuds, l'élément est répliqué sur tous les autres nœuds du cluster.

L'élément reste dans le cache d'éléments DAX, conformément au paramètre de time-to-live (TTL) et à l'algorithme LRU (moins récemment utilisé) définis pour le cache. Pour de plus amples informations, veuillez consulter Fonctionnement de DAX.

Toutefois, au cours de cette période, DAX ne relit pas l'élément à partir de DynamoDB. Si quelqu'un d'autre met à jour l'élément à l'aide d'un client DynamoDB, en ignorant totalement DAX, une demande `GetItem` effectuée à l'aide du client DAX génère des résultats différents de ceux que génère la même demande `GetItem` effectuée à l'aide du client DynamoDB. Dans ce scénario, DAX et DynamoDB contiennent des valeurs incohérentes pour la même clé tant que le time-to-live (TTL) de l'élément DAX n'a pas expiré.

Si une application modifie les données dans une table DynamoDB sous-jacente sans passer par DAX, l'application doit anticiper et tolérer les incohérences de données susceptibles d'apparaître.

> **Note**
>
> Outre la demande `GetItem`, le client DAX prend en charge les demandes `BatchGetItem`. Les demandes `BatchGetItem` encapsulant essentiellement une ou plusieurs autres demandes `GetItem`, DAX traite chacune de celles-ci comme une seule et même opération `GetItem`.

Cohérence des écritures

DAX est un cache d'écriture simultanée qui simplifie le processus de maintien de la cohérence entre le cache d'éléments DAX et les tables DynamoDB sous-jacentes.

Le client `PutItem` prend en charge les mêmes opérations d'API d'écriture que DynamoDB (, `UpdateItem`, `DeleteItem`, `BatchWriteItem` et `TransactWriteItems`). Lorsque vous utilisez ces opérations avec le client DAX, les éléments sont modifiés tant dans DAX que dans DynamoDB. DAX met à jour les éléments dans son cache d'élément, quelle que soit la valeur de time-to-live (TTL) de ces éléments.

Par exemple, supposons que vous émettez une demande `GetItem` à partir du client DAX pour lire un élément de la table `ProductCatalog`. (La clé de partition est `Id` ; il n'y a pas de clé de tri.) Vous extrayez l'élément dont `Id` a la valeur `101`. La valeur `QuantityOnHand` de cet élément est `42`. DAX stocke l'élément dans son cache d'éléments avec une TTL spécifique. Pour cet exemple, supposons que la durée de vie soit 10 minutes. Trois minutes plus tard, une autre application utilise le client DAX pour mettre à jour le même élément. La valeur `QuantityOnHand` de ce dernier est désormais `41`. En supposant que l'élément ne soit pas à nouveau mis à jour, les lectures de ce même élément qui se produisent au cours des dix minutes qui suivent retournent la valeur mise en cache de`QuantityOnHand` (soit `41`).

Comment DAX traite les écritures

DAX est destiné aux applications qui requièrent des lectures hautement performantes. En tant que cache d'écriture simultanée, DAX transmet vos écritures à DynamoDB de manière synchrone, puis réplique automatiquement et de manière asynchrone les mises à jour qui en résultent vers votre cache d'éléments dans tous les nœuds du cluster. Vous n'avez pas besoin de gérer une logique d'invalidation du cache, car DAX le fait à votre place.

DAX prend en charge les opérations d'écriture suivantes : `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` et `TransactWriteItems`.

Lorsque vous envoyez une demande `PutItem`, `UpdateItem`, `DeleteItem` ou `BatchWriteItem` à DAX, voici ce qui se produit :

- DAX envoie la demande à DynamoDB.
- DynamoDB répond à DAX, confirmant que l'écriture a abouti.
- DAX écrit l'élément dans son cache d'éléments.
- DAX indique au demandeur que l'opération a réussi.

Lorsque vous envoyez une demande `TransactWriteItems` à DAX, voici ce qui se produit :

- DAX envoie la demande à DynamoDB.
- DynamoDB répond à DAX, confirmant que la transaction est terminée.
- DAX indique au demandeur que l'opération a réussi.
- En arrière-plan, DAX effectue une demande `TransactGetItems` pour chaque élément dans la demande `TransactWriteItems`, afin de le stocker dans le cache d'éléments. `TransactGetItems` est utilisé pour assurer l'isolation sérialisable.

En cas d'échec de l'écriture sur DynamoDB pour une raison quelconque, y compris une limitation, l'élément n'est pas mis en cache dans DAX : L'exception relative à la défaillance est retournée au demandeur. Cela garantit que les données ne sont écrites dans le cache DAX que si elles ont préalablement écrites avec succès dans DynamoDB.

> **Note**
>
> Chaque écriture sur DAX modifie l'état du cache d'éléments. Cependant, les écritures sur le cache d'élément ne modifie pas le cache de requête. (Le cache d'éléments et le cache de requêtes DAX ayant des fonctions différentes, ils sont indépendants l'un de l'autre.)

Comportement du cache de requêtes DAX

DAX met en cache les résultats des demandes `Query` et `Scan` dans son cache de requêtes. Cependant, ces résultats n'affectent pas du tout le cache d'élément. Lorsque votre application émet

une demande `Query` ou `Scan` avec DAX, l'ensemble de résultats est enregistré dans le cache de requêtes, non dans le cache d'éléments. Vous ne pouvez pas solliciter le cache d'élément en effectuant une opération `Scan`, car le cache d'élément et le cache de requête sont des entités distinctes.

Cohérence de query-update-query

Les mises à jour du cache d'éléments ou de la table DynamoDB sous-jacente n'ont pas pour effet d'invalider ou de modifier les résultats stockés dans le cache de requêtes.

Pour bien comprendre cette distinction, prenez en compte le scénario suivant. Une application utilise la table `DocumentRevisions`, qui a `DocId` comme clé de partition et `RevisionNumber` comme clé de tri.

1. Un client émet une demande `Query` pour `DocId 101`, pour tous les éléments dont le `RevisionNumber` est supérieur ou égal à `5`. DAX stocke l'ensemble de résultats dans le cache de requêtes et le renvoie à l'utilisateur.
2. Le client émet une demande `PutItem` pour `DocId 101` avec `RevisionNumber` ayant la valeur `20`.
3. Le client émet la même opération `Query` que celle décrite à l'étape 1 (`DocId 101` et `RevisionNumber >= 5`).

Dans ce scénario, le jeu de résultats mis en cache pour l'opération `Query` émise à l'étape 3 est identique à celui qui a été mis en cache à l'étape 1. Cela est dû au fait que DAX n'invalide pas les ensembles de résultats des demandes `Query` ou `Scan` en fonction de mises d'éléments individuels. L'opération `PutItem` de l'étape 2 n'apparaît dans le cache de requêtes DAX qu'à l'expiration du time-to-live (TTL) de la demande `Query`.

Votre application doit prendre en compte la valeur de durée de vie (TTL) du cache de requête, ainsi que le temps durant lequel votre application peut tolérer une incohérence de résultats entre le cache de requête et le cache d'élément.

Lectures transactionnelles à cohérence forte

Pour effectuer une demande `GetItem`, `BatchGetItem`, `Query` ou `Scan` fortement cohérence, vous devez définir le paramètre `ConsistentRead` sur true. DAX transmet des demandes de lecture fortement cohérentes à DynamoDB. Sitôt la réponse de DynamoDB reçue, DAX renvoie les résultats au client sans toutefois les mettre en cache. DAX ne peut pas assurer les lectures

fortement cohérentes, car il n'est pas étroitement couplé à DynamoDB. C'est pourquoi, toute lecture subséquente à partir de DAX doit être une lecture éventuellement cohérente. Et toute lecture fortement cohérente subséquente doit être transmise à DynamoDB.

DAX gère les demandes `TransactGetItems` de la même façon que les lectures fortement cohérentes. DAX transmet toutes les demandes `TransactGetItems` à DynamoDB. Sitôt la réponse de DynamoDB reçue, DAX renvoie les résultats au client, mais ne les mets pas en cache.

Mise en cache négative

DAX prend en charge les entrées de cache négatives, tant dans le cache d'éléments que dans le cache de requêtes. Une entrée de cache négative se produit lorsque DAX ne trouve pas les éléments demandés dans une table DynamoDB sous-jacente. Au lieu de générer une erreur, DAX met en cache un résultat vide et le renvoie à l'utilisateur.

Par exemple, supposons qu'une application envoie une demande `GetItem` à un cluster DAX et que le cache d'éléments DAX ne contient aucun élément correspondant. DAX est alors contraint à lire l'élément correspondant dans la table DynamoDB sous-jacente. Si l'élément n'existe pas dans DynamoDB, DAX stocke un élément vide dans son cache d'éléments avant de renvoyer celui-ci à l'application. Supposons maintenant que l'application envoie un autre demande `GetItem` pour le même élément. DAX trouve l'élément vide dans le cache d'éléments et le renvoie immédiatement à l'application. À aucun moment il ne consulte DynamoDB.

Il reste une entrée de cache négative dans le cache d'éléments DAX jusqu'à ce que la TTL de l'élément expire, que son LRU soit appelé ou que l'élément soit modifié à l'aide de demandes `PutItem`, `UpdateItem` ou `DeleteItem`.

Le cache de requêtes DAX gère les résultats de cache négatifs de la même façon. Si une application effectue une opération `Query` ou `Scan` et que le cache de requêtes DAX ne contient pas de résultat mis en cache, DAX envoie la demande à DynamoDB. S'il n'existe pas d'élément correspondant dans l'ensemble de résultats, DAX stocke un ensemble de résultats vide dans le cache de requêtes, puis le renvoie à l'application. Les demandes `Query` ou `Scan` ultérieures génèrent le même jeu de résultats (vide) tant que le jeu de résultats n'a pas expiré.

Politiques pour les écritures

Le comportement d'écriture simultanée de DAX est adapté pour un grand nombre de modèles d'application. Toutefois, le modèle d'écriture simultanée peut ne pas convenir à certains modèles d'application.

Pour les applications sensibles à la latence, l'écriture via DAX génère un tronçon de réseau supplémentaire. Par conséquent, une écriture dans DAX est un peu plus lente qu'une écriture directe dans DynamoDB. Si votre application est sensible à la latence d'écriture, vous pouvez réduire celle-ci en écrivant directement dans DynamoDB. Pour de plus amples informations, veuillez consulter Écriture directe.

Pour les applications qui génèrent beaucoup d'écritures (comme celles qui procèdent à des chargements de données en bloc), il n'est peut-être pas souhaitable de passer par DAX pour écrire toutes les données, car l'application ne lit qu'un faible pourcentage de celles-ci. Lorsque vous écrivez de grandes quantités de données via DAX, celui-ci doit appeler son algorithme LRU afin de ménager de l'espace dans le cache pour les nouveaux éléments à lire. Cela a pour effet de diminuer l'efficacité de DAX en tant que cache de lecture.

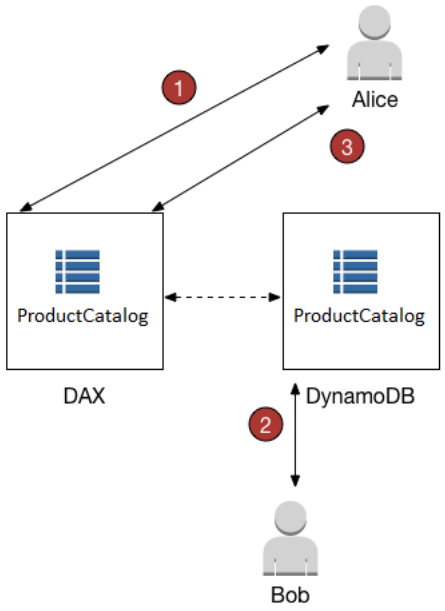
Lorsque vous écrivez un élément dans DAX, l'état du cache d'éléments change pour accueillir le nouvel élément. (Par exemple, DAX peut être amené à expulser les données les plus anciennes du cache d'éléments afin de ménager de l'espace pour le nouvel élément.) Le nouvel élément reste dans le cache d'élément, selon l'algorithme LRU et le paramètre de durée de vie (TTL) du cache. Tant que l'élément reste dans le cache d'éléments, DAX ne le relit pas dans DynamoDB.

Écriture simultanée

Le cache d'éléments DAX implémente une politique d'écriture simultanée. Pour de plus amples informations, veuillez consulter Comment DAX traite les écritures.

Lorsque vous écrivez un élément, DAX vérifie que l'élément mis en cache est synchronisé avec l'élément tel qu'il existe dans DynamoDB. Cela est utile pour les applications qui ont besoin de relire un élément de suite après l'avoir écrit. Toutefois, si d'autres applications écrivent directement dans une table DynamoDB, l'élément figurant dans le cache d'éléments DAX n'est plus synchronisé avec DynamoDB.

Pour illustrer cela, prenons l'exemple de deux utilisateurs (Alice et Bob) qui utilisent la table`ProductCatalog`. Alice accède à la table via DAX, mais Bob contourne DAX et accède à la table directement dans DynamoDB.



1. Alice met à jour un élément dans la table `ProductCatalog`. DAX transmet la demande à DynamoDB et la mise à jour aboutit. DAX écrit ensuite l'élément dans son cache d'éléments et renvoie une réponse positive à Alice. À partir de là, tant que l'élément n'est pas expulsé du cache, tout utilisateur qui lit l'élément à partir de DAX voit l'élément avec la mise à jour d'Alice.
2. Quelques instants plus tard, Bob met à jour le même élément `ProductCatalog` qu'Alice a écrit. Cependant, Bob met à jour l'élément directement dans DynamoDB. DAX n'actualise pas automatiquement son cache d'éléments en réponse aux mises à jour via DynamoDB. Par conséquent, les utilisateurs de DAX ne voient pas la mise à jour de Bob.
3. Alice lit à nouveau l'élément à partir de DAX. Sachant que l'élément se trouve dans le cache d'éléments, DAX le renvoie à Alice sans accéder à la table DynamoDB.

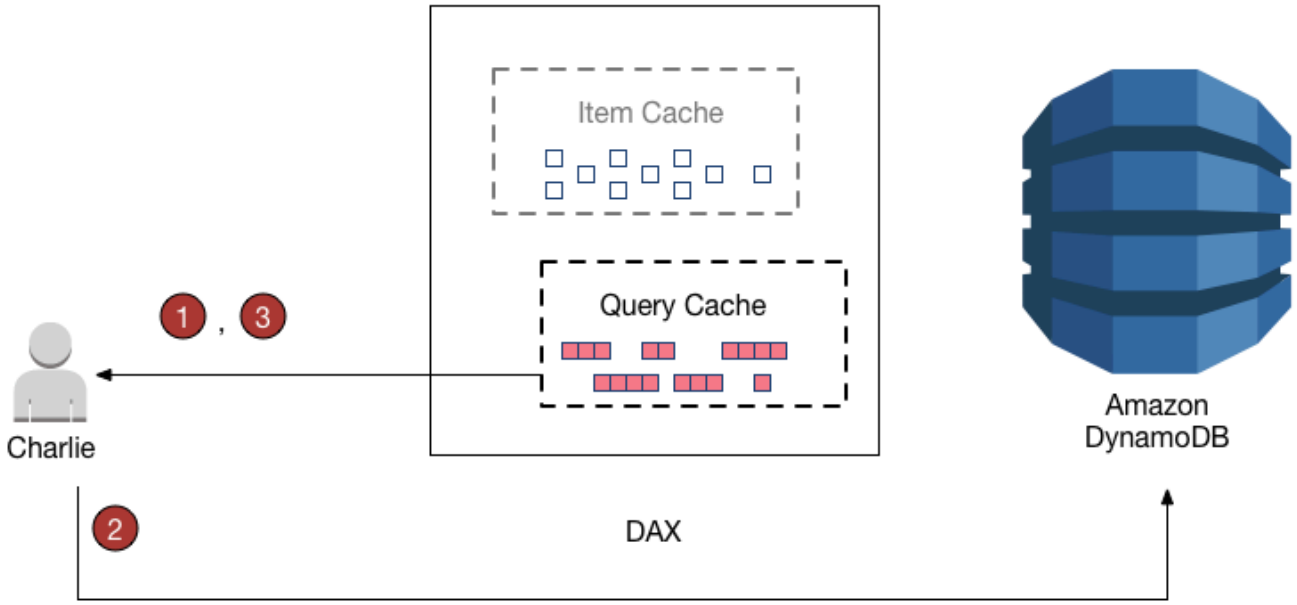
Dans ce scénario, Alice et Bob voient des représentations différentes du même élément `ProductCatalog`. Ce sera le cas tant que DAX n'aura pas expulsé l'élément du cache d'éléments ou qu'un autre utilisateur n'aura pas remis à jour ce même élément à partir de DAX.

Écriture directe

Si votre application a besoin d'écrire de grandes quantités de données (par exemple, pour charger des données en bloc), il peut être judicieux de contourner DAX et d'écrire les données directement dans DynamoDB. Une telle stratégie d'écriture non allouée réduit la latence d'écriture. Cependant, le cache d'éléments ne reste pas synchronisé avec les données dans DynamoDB.

Si vous décidez d'utiliser une politique d'écriture simultanée, ne perdez pas de vue que DAX remplit son cache d'éléments chaque fois que des applications utilisent le client DAX pour lire des données. Dans certains cas, vous pouvez en tirer des avantages, car seules les données les plus fréquemment lues sont mises à en cache (et non les données les plus fréquemment écrites).

Prenons, par exemple, le cas d'un utilisateur (Charlie) qui souhaite travailler avec une autre table, la table `GameScores`, en utilisant DAX. Sachant que la clé de partition de `GameScores` est `UserId`, tous les scores de Charlie ont le même `UserId`.



1. Charlie souhaitant récupérer tous ses scores. Il envoie une demande `Query` à DAX. Présumant que cette requête n'a pas été émise auparavant, DAX l'achemine vers DynamoDB pour traitement. Il stocke les résultats dans le cache de requêtes DAX avant de les renvoyer à Charlie. Le jeu de résultats reste disponible dans le cache de requête jusqu'à ce qu'il soit expulsé.
2. Supposons maintenant que Charlie joue au jeu Meteor Blasters et qu'il atteint un score élevé. Charlie envoie une demande `UpdateItem` à DynamoDB, modifiant ainsi un élément dans la table `GameScores`.
3. Enfin, Charlie décide de réexécuter son opération `Query` précédente pour extraire toutes ses données de `GameScores`. Charlie ne trouve pas le score élevé qu'il a obtenu à Meteor Blasters dans les résultats. Cela est dû au fait que les résultats de la requête proviennent du cache de requête, et non du cache d'élément. Les deux caches étant indépendants l'un de l'autre, toute modification dans un cache ne se répercute pas dans l'autre cache.

DAX n'actualise pas les ensembles de résultats dans le cache de requêtes avec les données les plus récentes de DynamoDB. Chaque jeu de résultats du cache de requête correspond au moment où l'opération `Query` ou `Scan` a été effectuée. Par conséquent, les résultats de l'opération `Query` de Charlie ne reflètent pas son opération `PutItem`. C'est le cas jusqu'à ce que DAX expulse l'ensemble de résultats du cache de requêtes.

Développement avec le client DynamoDB Accelerator (DAX)

Pour utiliser DAX à partir d'une application, vous utilisez le client DAX pour votre langage de programmation. Le client DAX est conçu pour minimiser la perturbation de vos applications Amazon DynamoDB existantes. Seules quelques modifications de code simples sont nécessaires.

> **Note**
>
> Des clients DAX pour différents langages de programmation sont disponibles sur le site suivant :
>
> - http://dax-sdk.s3 - website-us-west -2.amazonaws.com

Cette section explique comment lancer une EC2 instance Amazon dans votre Amazon VPC par défaut, vous connecter à l'instance et exécuter un exemple d'application. Elle fournit également des informations sur la façon de modifier votre application existante afin qu'elle puisse utiliser votre cluster DAX.

Rubriques

- Didacticiel : exécution d'un exemple d'application avec DynamoDB Accelerator (DAX)
- Modification d'une application existante pour utiliser DAX

Didacticiel : exécution d'un exemple d'application avec DynamoDB Accelerator (DAX)

Ce didacticiel explique comment lancer une EC2 instance Amazon dans votre cloud privé virtuel (VPC) par défaut, vous connecter à l'instance et exécuter un exemple d'application utilisant Amazon DynamoDB Accelerator (DAX).

> **Note**
>
> Pour suivre de didacticiel, vous devez disposer d'un cluster DAX s'exécutant dans votre VPC par défaut. Si vous n'avez pas créé de cluster DAX, consultez Création d'un cluster DAX pour obtenir des instructions.

Rubriques

- Étape 1 : Lancer une EC2 instance Amazon
- Étape 2 : Création d'un utilisateur et d'une stratégie
- Étape 3 : Configuration d'une EC2 instance Amazon
- Étape 4 : exécuter un exemple d'application

Étape 1 : Lancer une EC2 instance Amazon

Lorsque votre cluster Amazon DynamoDB Accelerator (DAX) est disponible, vous pouvez lancer une instance Amazon dans votre EC2 Amazon Virtual Private Cloud (Amazon VPC) par défaut. Vous pouvez ensuite installer et exécuter le logiciel client DAX sur cette instance.

Pour lancer une EC2 instance

1. Connectez-vous à la EC2 console Amazon AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/ec2/.
2. Choisissez Lancer une instance, puis effectuez les actions suivantes :

   Étape 1 : Sélection d'une Amazon Machine Image (AMI)

   1. Dans la liste AMIs, recherchez l'AMI Amazon Linux, puis choisissez Select.

   Étape 2 : Choisir un type d'instance

   1. En haut de la liste des types d'instance, choisissez t2.micro.
   2. Sélectionnez Next: Configure Instance Details.

   Étape 3 : Configurer les détails de l'instance

   1. Pour Network (Réseau), choisissez votre VPC par défaut.

2. Choisissez Next: Add Storage (Suivant : Ajouter le stockage).

Étape 4 : Ajouter du stockage

1. Ignorez cette étape en choisissant Suivant : Ajouter des balises.

Étape 5 : Ajouter des balises

1. Ignorez cette étape en choisissant Next: Configure Security Group.

Étape 6 : Configurer un groupe de sécurité

1. Choisissez Select an existing security group.
2. Dans la liste des groupes de sécurité, choisissez default. Il s'agit du groupe de sécurité par défaut pour votre VPC.
3. Choisissez Next: Review and Launch.

Étape 7 : Examiner le lancement de l'instance

1. Choisissez Lancer.

3. Dans la fenêtre Select an existing key pair or create a new key pair, sélectionnez l'une des options suivantes :

   - Si vous ne possédez pas de paire de EC2 clés Amazon, choisissez Create a new key pair et suivez les instructions. Vous êtes invité à télécharger un fichier de clé privée (fichier `.pem`). Vous aurez besoin de ce fichier ultérieurement lorsque vous vous connecterez à votre EC2 instance Amazon.
   - Si vous possédez déjà une paire de EC2 clés Amazon, rendez-vous sur Sélectionner une paire de clés et choisissez votre paire de clés dans la liste. Le fichier de clé privée (`.pem`fichier) doit déjà être disponible pour vous connecter à votre EC2 instance Amazon.

4. Lorsque vous aurez configuré votre paire de clés, choisissez Lancer les instances.
5. Dans le volet de navigation de la console, choisissez EC2 Dashboard, puis choisissez l'instance que vous avez lancée. Dans le volet inférieur, sous l'onglet Description, recherchez le DNS public de votre instance, par exemple, `ec2-11-22-33-44.us-`

`west-2.compute.amazonaws.com`. Notez ce nom de DNS public, car vous en aurez besoin pour Étape 3 : Configuration d'une EC2 instance Amazon.

> **Note**
>
> Il faut quelques minutes pour que votre EC2 instance Amazon soit disponible. Pendant ce temps, vous pouvez passer à Étape 2 : Création d’un utilisateur et d’une stratégie et suivre les instructions qui y sont fournies.

Étape 2 : Création d’un utilisateur et d’une stratégie

Au cours de cette étape, vous créez un utilisateur doté d'une politique qui accorde l'accès à votre cluster Amazon DynamoDB Accelerator (DAX) et à DynamoDB à l'aide de.Gestion des identités et des accès AWS Vous pouvez ensuite exécuter des applications qui interagissent avec votre cluster DAX.

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d’inscription, vous recevrez un appel téléphonique ou un SMS et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l’utilisateur racine a accès à l’ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l’utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d’un utilisateur doté d’un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWSAWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l’aide pour vous connecter en utilisant l’utilisateur racine, consultez Connexion en tant qu’utilisateur racine dans le Guide de l’utilisateur Connexion à AWS.
2. Activez l’authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, consultez la section Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d’un utilisateur doté d’un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d’AWS IAM Identity Center dans le Guide de l’utilisateur AWS IAM Identity Center.
2. Dans IAM Identity Center, octroyez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu’utilisateur doté d’un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l’URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l’utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations qui respecte la bonne pratique consistant à appliquer les autorisations de moindre privilège.

   Pour obtenir des instructions, consultez Création d'un ensemble d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center.
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l'utilisateur AWS IAM Identity Center.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center.
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Pour utiliser l'éditeur de politique JSON afin de créer une politique

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Dans le panneau de navigation de gauche, sélectionnez Policies (Politiques).

Si vous sélectionnez Politiques pour la première fois, la page Bienvenue dans les politiques gérées s'affiche. Sélectionnez Mise en route.

3. En haut de la page, sélectionnez Créer une politique.
4. Dans la section Éditeur de politique, choisissez l'option JSON.
5. Saisissez ou collez un document de politique JSON. Pour de plus amples informations sur le langage de la stratégie IAM, consultez la référence de politique JSON IAM.
6. Résolvez les avertissements de sécurité, les erreurs ou les avertissements généraux générés durant la validation de la politique, puis choisissez Suivant.

> **Note**
>
> Vous pouvez basculer à tout moment entre les options des éditeurs visuel et JSON. Toutefois, si vous apportez des modifications ou si vous choisissez Suivant dans l'éditeur visuel, IAM peut restructurer votre politique afin de l'optimiser pour l'éditeur visuel. Pour plus d'informations, consultez la page Restructuration de politique dans le Guide de l'utilisateur IAM.

7. (Facultatif) Lorsque vous créez ou modifiez une politique dans le AWS Management Console, vous pouvez générer un modèle de stratégie JSON ou YAML que vous pouvez utiliser dans les CloudFormation modèles.

   Pour ce faire, dans l'éditeur de politiques, sélectionnez Actions, puis sélectionnez Générer CloudFormation un modèle. Pour en savoir plus CloudFormation, consultez la référence aux types de Gestion des identités et des accès AWS ressources dans le Guide de AWS CloudFormation l'utilisateur.
8. Lorsque vous avez fini d'ajouter des autorisations à la politique, choisissez Suivant.
9. Sur la page Vérifier et créer, tapez un Nom de politique et une Description (facultative) pour la politique que vous créez. Vérifiez les Autorisations définies dans cette politique pour voir les autorisations accordées par votre politique.
10. (Facultatif) Ajoutez des métadonnées à la politique en associant les balises sous forme de paires clé-valeur. Pour plus d'informations sur l'utilisation des balises dans IAM, consultez la section Balises pour les Gestion des identités et des accès AWS ressources dans le Guide de l'utilisateur d'IAM.
11. Choisissez Create policy (Créer une politique) pour enregistrer votre nouvelle politique.

Document de politique – Copiez et collez le document suivant pour créer la politique JSON.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Étape 3 : Configuration d'une EC2 instance Amazon

Lorsque votre EC2 instance Amazon est disponible, vous pouvez vous connecter à l'instance et préparer son utilisation.

> **Note**
>
> Les étapes suivantes supposent que vous vous connectez à votre EC2 instance Amazon depuis un ordinateur exécutant Linux. Pour d'autres méthodes de connexion, consultez Connect to Your Linux Instance dans le guide de EC2 l'utilisateur Amazon.

Pour configurer l' EC2 instance

1. Ouvrez la EC2 console Amazon à l'adresse https://console.aws.amazon.com/ec2/.
2. Utilisez la `ssh` commande pour vous connecter à votre EC2 instance Amazon, comme illustré dans l'exemple suivant.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Vous devez spécifier votre fichier de clé privée (fichier `.pem`) et le nom DNS public de votre instance. (Consultez Étape 1 : Lancer une EC2 instance Amazon.)

   L'ID de connexion est `ec2-user`. Aucun mot de passe n'est requis.
3. Une fois connecté à votre EC2 instance, configurez vos AWS informations d'identification comme indiqué ci-dessous. Entrez l'ID de votre clé d'AWS accès et votre clé secrète (deÉtape 2 : Création d'un utilisateur et d'une stratégie), et définissez le nom de région par défaut sur votre région actuelle. (Dans l'exemple suivant, le nom de la région par défaut est `us-west-2`.)

   ```
   aws configure AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLEAWS Secret Access Key
    [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Après avoir lancé et configuré votre EC2 instance Amazon, vous pouvez tester les fonctionnalités de DAX à l'aide de l'un des exemples d'applications disponibles. Pour de plus amples informations, veuillez consulter Étape 4 : exécuter un exemple d'application.

Étape 4 : exécuter un exemple d'application

Pour vous aider à tester les fonctionnalités d'Amazon DynamoDB Accelerator (DAX), vous pouvez exécuter l'un des exemples d'applications disponibles sur votre instance Amazon. EC2

Rubriques

- Node.js et DAX
- Kit SDK DAX pour Go
- Java et DAX
- .NET et DAX
- Python et DAX

Node.js et DAX

Configuration du client par défaut pour Node.js

Lors de la configuration du client DAX JavaScript SDK, vous pouvez personnaliser divers paramètres pour optimiser les performances, la gestion des connexions et la résilience aux erreurs. Le tableau suivant décrit les paramètres de configuration par défaut qui contrôlent la manière dont votre client interagit avec le cluster DAX, notamment les valeurs de délai d'expiration, les mécanismes de nouvelle tentative, la gestion des informations d'identification et les options de surveillance de l'état de santé. Pour plus d'informations, consultez Dynamo DBClient Operations.

Paramètres par défaut du kit SDK JS DAX

Paramètre	Type	Description
`region` facultatif	`string`	Région AWSÀ utiliser pour le client DAX (exemple : « us-east-1 »). Il s'agit d'un paramètre obligatoire s'il n'est pas fourni par la variable d'environnement.
`endpoint` obligatoire	`string`	Point de terminaison du cluster auquel le kit SDK se connecte. Exemples : Non crypté — .region.amazonaws.com dax-cluster-name Chiffré : dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`requestTimeout` Par défaut : 6 000 ms	`number`	Cela définit le temps maximum pendant lequel le client attend une réponse de DAX.

Paramètre	Type	Description
`writeRetries` Par défaut : 1	`number`	Nombre de nouvelles tentatives pour des demandes d'écriture qui ont échoué.
`readRetries` Par défaut : 1	`number`	Nombre de nouvelles tentatives pour des demandes de lecture qui ont échoué.
`maxRetries` Par défaut : 1	`number`	Nombre maximal de nouvelles tentatives en cas d'échec des demandes. S' readRetries/writeRetries ils sont définis, la configuration définie dans ReadRetries et WriteRetries a priorité sur MaxRetries.
`connectTimeout` Par défaut : 10 000 ms	`number`	Délai (en millisecondes) pour établir une connexion à l'un des nœuds du cluster.
`maxConcurrentConnections` Par défaut : 100	`number`	Limite le nombre total de connexions simultanées qu'une instance client peut créer par nœud dans un cluster DAX.

Paramètre	Type	Description
`maxRetryDelay` Par défaut : 7 000 ms	`number`	Lorsque le serveur DAX indique qu'une récupération est nécessaire en définissant l'indicateur `waitForRecoveryBeforeRetrying` sur true, le client fait une pause avant de réessayer. Pendant ces périodes de récupération, le paramètre `maxRetryDelay` détermine le temps d'attente maximal entre les tentatives. Cette configuration spécifique à la récupération s'applique uniquement lorsque le serveur DAX est en mode récupération. Pour tous les autres scénarios, le comportement des nouvelles tentatives suit l'un des deux modèles suivants : soit un délai exponentiel basé sur le nombre de tentatives (régi par les paramètres `writeRetries`, `readRetries` ou `maxRetries`), soit une nouvelle tentative immédiate en fonction du type d'exception.

Paramètre	Type	Description
`credentials` facultatif	AwsCredentialIdentity | AwsCredentialIdentityProvider	Les AWS informations d'identification à utiliser pour authentifier les demandes. Cela peut être fourni sous forme de AwsCredentialIdentity ou de AwsCredentialIdentityProvider. Si ces paramètres ne sont pas fournis, le kit SDK AWS utilisera automatiquement la chaîne du fournisseur d'informations d'identification par défaut. Exemple : `{ accessKeyId: 'AKIA... ' , secretAccessKey : ' ... ', SessionToken :'... '} `* @default Utilise la chaîne de fournisseurs d'informations d'identification AWS par défaut.
`healthCheckInterval` Par défaut : 5 000 ms	`number`	Intervalle (en millisecondes) entre les surveillances de l'état du cluster. Un intervalle inférieur permet d'effectuer des surveillances plus fréquentes.
`healthCheckTimeout` Par défaut : 1 000 ms	`number`	Délai (en millisecondes) nécessaire à l'exécution de la surveillance de l'état.

Paramètre	Type	Description
`skipHostnameVerification` Par défaut : false	`boolean`	Ignore la vérification du nom d'hôte des connexions TLS. Cela n'a aucun impact sur les clusters non chiffrés. Par défaut, la vérification du nom d'hôte est effectuée. Si vous définissez ce paramètre sur True, la vérification sera ignorée. Assurez-vous de bien comprendre les conséquences de sa désactivation, à savoir l'impossibilité d'authentifier le cluster auquel vous vous connectez.
`unhealthyConsecutiveErrorCount` Par défaut : 5	`number`	Définit le nombre d'erreurs consécutives nécessaires pour signaler un nœud défectueux pendant l'intervalle de surveillance de l'état.
`clusterUpdateInterval` Par défaut : 4 000 ms	`number`	Renvoie l'intervalle entre les interrogations des membres du cluster pour les modifications d'appartenance.
`clusterUpdateThreshold` Par défaut : 125	`number`	Renvoie le seuil en dessous duquel le cluster ne sera pas interrogé pour les modifications d'appartenance.

Paramètre	Type	Description
`credentailProvider` facultatif | null par défaut	AwsCredentialIdentityProvider	Fournisseur défini par l'utilisateur pour les AWS informations d'identification utilisées pour authentifier les demandes adressées au DAX.

Configuration de pagination pour DaxDocument

Name	Type	Détail
`client`	DaxDocument	Instance de DaxDocument type.
`pageSize`	number	Détermine le nombre d'éléments par page.
`startingToken` Facultatif	any	LastEvaluatedKey La réponse précédente peut être utilisée pour les demandes suivantes.

Pour plus d'informations sur l'utilisation de la pagination, consultez the section called "TryDax.js".

Migration vers le kit SDK Node.js DAX V3

Ce guide de migration vous aidera à effectuer la transition de vos applications Node.js DAX existantes. Le nouveau kit SDK nécessite Node.js 18 ou une version ultérieure et introduit plusieurs modifications importantes dans la manière dont vous allez structurer votre code DynamoDB Accelerator. Ce guide vous expliquera les principales différences, notamment les modifications de syntaxe, les nouvelles méthodes d'importation et les modèles de programmation asynchrone mis à jour.

Utilisation de Node.js DAX V2

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');

var region = "us-west-2";
```

```
AWS.config.update({
  region: region,
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

Utilisation de Node.js DAX V3

Concernant l'utilisation de DAX Node.js V3, la version 18 ou supérieure est la version préférée. Pour passer à Node 18, utilisez les éléments suivants :

```
// Import AWS DAX V3
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';

// Import AWS SDK V3 DynamoDBDocument ~ DocumentClient in V2
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
import { DynamoDBClient } from '@aws-sdk/client-dyanmodb';
```

```
// Create DynamoDBDocument
var client = DynamoDBDocument.from(new DynamoDB({region: 'us-west-2'});

// Override DynamoDBDocument Client with DaxDocument
if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
  });
}

var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
// Dax Shifted it's API Calls to await/promise
try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err)
}
```

Le kit SDK DAX pour Node.js v3.x est compatible avec l'AWS SDK pour Node.js v3.x. Le kit SDK DAX pour Node.js v3.x prend en charge l'utilisation de clients regroupés. Notez que DAX ne prend pas en charge la création de clients bone. Pour plus de détails sur les fonctionnalités non prises en charge, consultez the section called "Fonctionnalités non équivalentes à celles du AWS SDK V3".

Suivez ces étapes pour exécuter l'exemple d'application Node.js sur votre EC2 instance Amazon.

Pour exécuter l'exemple Node.js pour DAX

1. Configurez Node.js sur votre EC2 instance Amazon, comme suit :

   a. Installez le gestionnaire de version de nœud (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Utilisez nvm pour installer Node.js.

```
nvm install 18
```

c. Utilisez nvm pour l'utilisation de Node 18

```
nvm use 18
```

d. Testez que Node.js est installé et fonctionne correctement.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Le message suivant doit s'afficher.

```
Running Node.js v18.x.x
```

2. Installez le client DaxDocument Node.js à l'aide du gestionnaire de packages de nœuds (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

TryDaxexemple de code

Pour évaluer les avantages en matière de performances de DynamoDB Accelerator (DAX), procédez comme suit pour exécuter un exemple de test qui compare les temps de lecture entre DynamoDB standard et un cluster DAX.

1. Après avoir configuré votre espace de travail et installé `lib-dax` en tant que dépendance, copiez the section called "TryDax.js" dans votre projet.
2. Exécutez le programme sur votre cluster DAX. Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

   - Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - À l'aide de AWS CLI— Entrez la commande suivante.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

3. À présent, exécutez le programme en spécifiant le point de terminaison de cluster comme paramètre de ligne de commande.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Vous devez voir des résultats similaires à ce qui suit :

```
Attempting to create table; please wait...
Successfully created table. Table status: ACTIVE
Writing data to the table...
Writing 20 items for partition key:  1
Writing 20 items for partition key:  2
Writing 20 items for partition key:  3
...
Running GetItem Test
        Total time: 153555.10 µs - Avg time: 383.89 µs
        Total time: 44679.96 µs - Avg time: 111.70 µs
        Total time: 36885.86 µs - Avg time: 92.21 µs
        Total time: 32467.25 µs - Avg time: 81.17 µs
        Total time: 32202.60 µs - Avg time: 80.51 µs
Running Query Test
        Total time: 14869.25 µs - Avg time: 2973.85 µs
        Total time: 3036.31 µs - Avg time: 607.26 µs
        Total time: 2468.92 µs - Avg time: 493.78 µs
        Total time: 2062.53 µs - Avg time: 412.51 µs
        Total time: 2178.22 µs - Avg time: 435.64 µs
Running Scan Test
        Total time: 2395.88 µs - Avg time: 479.18 µs
        Total time: 2207.16 µs - Avg time: 441.43 µs
        Total time: 2443.14 µs - Avg time: 488.63 µs
        Total time: 2038.24 µs - Avg time: 407.65 µs
        Total time: 1972.17 µs - Avg time: 394.43 µs
```

```
Running Pagination Test
Scan Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Query Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Attempting to delete table; please wait...
Successfully deleted table.
```

Notez les informations relatives à la durée, c'est-à-dire le nombre de microsecondes requis pour les tests `GetItem`, `Query` et `Scan`.

4. Dans ce cas, vous avez exécuté les programmes sur le cluster DAX. Vous allez maintenant réexécuter le programme mais, cette fois, sur DynamoDB.
5. À présent, réexécutez le programme mais, cette fois, sans l'URL du point de terminaison du cluster en tant que paramètre de ligne de commande.

```
node TryDax.js
```

Examinez la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX par rapport à DynamoDB.

Fonctionnalités non équivalentes à celles du AWS SDK V3

- Clients bone : Dax Node.js V3 ne prend pas en charge les clients élémentaires.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
const regularParams = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};
// The DynamoDB client supports the send operation.
const dynamoResult = await dynamoDBClient.send(new GetCommand(regularParams));

// However, the DaxDocument client does not support the send operation.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

const params = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};

// This will throw an error - send operation is not supported for DAX. Please refer
 to documentation.
const result = await daxClient.send(new GetCommand(params));
console.log(result);
```

- Pile d'intergiciel : Dax Node.js V3 ne prend pas en charge l'utilisation des fonctions d'intergiciel.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
// The DynamoDB client supports the middlewareStack.
dynamoDBClient.middlewareStack.add(
  (next, context) =>> async (args) => {
    console.log("Before operation:", args);
```

```
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);

// However, the DaxDocument client does not support the middlewareStack.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

// This will throw an error - custom middleware and middlewareStacks are not
 supported for DAX. Please refer to documentation.
daxClient.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);
```

TryDax.js

```
import { DynamoDB, waitUntilTableExists, waitUntilTableNotExists } from "@aws-sdk/
client-dynamodb";
import { DaxDocument, daxPaginateScan, daxPaginateQuery } from "@amazon-dax-sdk/lib-
dax";
import { DynamoDBDocument, paginateQuery, paginateScan } from "@aws-sdk/lib-dynamodb";

const region = "us-east-1";
const tableName = "TryDaxTable";
```

```
// Determine the client (DynamoDB or DAX)
let client = DynamoDBDocument.from(new DynamoDB({ region }));
if (process.argv.length > 2) {
  client = new DaxDocument({ region, endpoint: process.argv[2] });
}

// Function to create table
async function createTable() {
  const dynamodb = new DynamoDB({ region });
  const params = {
    TableName: tableName,
    KeySchema: [
      { AttributeName: "pk", KeyType: "HASH" },
      { AttributeName: "sk", KeyType: "RANGE" },
    ],
    AttributeDefinitions: [
      { AttributeName: "pk", AttributeType: "N" },
      { AttributeName: "sk", AttributeType: "N" },
    ],
    ProvisionedThroughput: { ReadCapacityUnits: 25, WriteCapacityUnits: 25 },
  };

  try {
    console.log("Attempting to create table; please wait...");
    await dynamodb.createTable(params);
    await waitUntilTableExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully created table. Table status: ACTIVE");
  } catch (err) {
    console.error("Error in table creation:", err);
  }
}

// Function to insert data
async function writeData() {
  console.log("Writing data to the table...");
  const someData = "X".repeat(10);
  for (let ipk = 1; ipk <= 20; ipk++) {
    console.log("Writing 20 items for partition key: ", ipk)
    for (let isk = 1; isk <= 20; isk++) {
      try {
        await client.put({ TableName: tableName, Item: { pk: ipk, sk: isk,
 someData } });
```

```
      } catch (err) {
        console.error("Error inserting data:", err);
      }
    }
  }
}

// Function to test GetItem
async function getItemTest() {
  console.log("Running GetItem Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let ipk = 1; ipk <= 20; ipk++) {
      for (let isk = 1; isk <= 20; isk++) {
        promises.push(client.get({ TableName: tableName, Key: { pk: ipk, sk: isk } }));
      }
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 400).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Query
async function queryTest() {
  console.log("Running Query Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.query(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
```

```
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Scan
async function scanTest() {
  console.log("Running Scan Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        FilterExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.scan(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Pagination
async function paginationTest() {
  console.log("Running Pagination Test");
  console.log("Scan Pagination");
  const scanParams = { TableName: tableName };
  const paginator = process.argv.length > 2 ? daxPaginateScan : paginateScan;
  for await (const page of paginator({ client, pageSize: 3 }, scanParams)) {
    console.log(page.Items);
  }

  console.log("Query Pagination");
  const queryParams = {
    TableName: tableName,
    KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
    ExpressionAttributeValues: { ":pkval": 1, ":skval1": 1, ":skval2": 10 },
```

```
  };
  const queryPaginator = process.argv.length > 2 ? daxPaginateQuery : paginateQuery;
  for await (const page of queryPaginator({ client, pageSize: 3 }, queryParams)) {
    console.log(page.Items);
  }
}

// Function to delete the table
async function deleteTable() {
  const dynamodb = new DynamoDB({ region });
  console.log("Attempting to delete table; please wait...")
  try {
    await dynamodb.deleteTable({ TableName: tableName });
    await waitUntilTableNotExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully deleted table.");
  } catch (err) {
    console.error("Error deleting table:", err);
  }
}

// Execute functions sequentially
(async function () {
  await createTable();
  await writeData();
  await getItemTest();
  await queryTest();
  await scanTest();
  await paginationTest();
  await deleteTable();
})();
```

Kit SDK DAX pour Go

Suivez cette procédure pour exécuter l'exemple d'application SDK Amazon DynamoDB Accelerator (DAX) pour Go sur votre instance Amazon. EC2

Pour exécuter l’exemple du kit SDK pour Go pour DAX

1. Configurez le SDK pour Go sur votre instance EC2 Amazon :

    a. Installez le langage de programmation Go (`Golang`).

```
sudo yum install -y golang
```

b. Testez que Golang est installé et fonctionne correctement.

```
go version
```

Un message comme celui-ci devrait s'afficher.

```
go version go1.23.4 linux/amd64
```

2. Installez l'exemple d'application Golang.

```
go get github.com/aws-samples/sample-aws-dax-go-v2
```

3. Exécutez les programmes Golang suivants. Le premier programme crée une table DynamoDB nommée `TryDaxGoTable`. Le deuxième programme écrit des données dans la table.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command put-item
```

4. Exécutez les programmes Golang suivants.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command scan
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

5. A l'étape précédente, vous avez exécuté les programmes par rapport au point de terminaison DynamoDB. A présent, réexécutez-les mais, cette fois, les opérations `GetItem`, `Query` et `Scan` sont traitées par votre cluster DAX.

   Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

   - Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - À l'aide de AWS CLI— Entrez la commande suivante.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   À présent, réexécutez les programmes, mais cette fois, spécifiez le point de terminaison du cluster en tant que paramètre de ligne de commande.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-batch-get -endpoint my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com:8111
```

Observez le reste de la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX qu'avec DynamoDB.

6. Exécutez le programme Golang suivant pour supprimer `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command delete-table
```

Fonctionnalités non équivalentes à celles de la AWS SDK pour Go V2

Middleware Stack — DAX Go V2 ne prend pas en charge l'utilisation de Middleware Stacks through. APIoptions Pour plus d'informations, voir Personnalisation des demandes du client AWS SDK pour Go v2 avec un intergiciel.

Exemple :

```
// Custom middleware implementation
type customSerializeMiddleware struct{}
// ID returns the identifier for the middleware
func (m *customSerializeMiddleware) ID() string {
    return "CustomMiddleware"
}
// HandleSerialize implements the serialize middleware handler
func (m *customSerializeMiddleware) HandleSerialize(
    ctx context.Context,
    in middleware.SerializeInput,
    next middleware.SerializeHandler,
```

```
) (
    out middleware.SerializeOutput,
    metadata middleware.Metadata,
    err error,
) {
    // Add your custom logic here before the request is serialized
    fmt.Printf("Executing custom middleware for request: %v\n", in)
    // Call the next handler in the middleware chain
    return next.HandleSerialize(ctx, in)
}

func executeGetItem(ctx context.Context) error {
    client, err := initItemClient(ctx)
    if err != nil {
        os.Stderr.WriteString(fmt.Sprintf("failed to initialize client: %v\n", err))
        return err
    }

    st := time.Now()
    for c := 0; c < iterations; c++ {
        for i := 0; i < pkMax; i++ {
            for j := 0; j < skMax; j++ {
                // Create key using attributevalue.Marshal for type safety
                pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
                if err != nil {
                    return fmt.Errorf("error marshaling pk: %v", err)
                }
                sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
                if err != nil {
                    return fmt.Errorf("error marshaling sk: %v", err)
                }
                key := map[string]types.AttributeValue{
                    "pk": pk,
                    "sk": sk,
                }
                in := &dynamodb.GetItemInput{
                    TableName: aws.String(table),
                    Key:       key,
                }

                // Custom middleware option
                customMiddleware := func(o *dynamodb.Options) {
                    o.APIOptions = append(o.APIOptions, func(stack *middleware.Stack)
 error {
```

```
                        // Add custom middleware to the stack
                        return stack.Serialize.Add(&customSerializeMiddleware{},
 middleware.After)
                    })
                }

                // Apply options to the GetItem call
                out, err := client.GetItem(ctx, in, customMiddleware)
                if err != nil {
                    return err
                }
                writeVerbose(out)
            }
        }
    }
    d := time.Since(st)
    os.Stdout.WriteString(fmt.Sprintf("Total Time: %v, Avg Time: %v\n", d, d/
iterations))
    return nil
}
```

Sortie :

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Configuration du client par défaut pour Go

Ce guide présente les options de configuration qui vous permettent d'optimiser les performances, la gestion des connexions et le comportement de journalisation de votre client DAX. En comprenant les paramètres par défaut et en sachant comment les personnaliser, vous pouvez optimiser l'interaction de votre application Go avec DAX.

Dans cette section

- Paramètres par défaut du client de kit SDK Go DAX
- Création de client

Paramètres par défaut du client de kit SDK Go DAX

Paramètre	Type	Description
`Region` obligatoire	`string`	Région AWSÀ utiliser pour le client DAX (exemple- 'us-east-1'). Il s'agit d'un paramètre obligatoire s'il n'est pas fourni par l'environnement.
`HostPorts` obligatoire	`[] string`	Liste des points de terminaison du cluster DAX auxquels le kit SDK se connecte. Par exemple : Non chiffré : dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com Chiffré : dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` Par défaut : 10	`number`	Nombre de tentatives de connexion simultanées. (Les connexions peuvent être en cours d'établissement simultanément.)
`ClusterUpdateThreshold` Par défaut : 125 * time.Millisecond	`time.Duration`	Durée minimale qui doit s'écouler entre les actualisations du cluster.
`ClusterUpdateInterval`	`time.Duration`	Intervalle selon lequel le client actualise automatiquement les informations du cluster DAX.

Paramètre	Type	Description
Par défaut : 4 * time.Second		
`IdleConnectionsReapDelay` Par défaut : 30 * time.Second	`time.Duration`	Intervalle selon lequel le client fermera les connexions inactives dans le client DAX.
`ClientHealthCheckInterval` Par défaut : 5 * time.Second	`time.Duration`	Intervalle selon lequel le client effectuera des surveillances de l'état sur les points de terminaison du cluster DAX.
`Credentials` default	`aws.CredentialsProvider`	Les AWS informations d'identification utilisées par le client DAX pour authentifier les demandes adressées au service DAX. Consultez Credentials and Credential Providers.
`DialContext` default	`func`	Fonction personnalisée utilisée par le client DAX pour établir des connexions au cluster DAX.

Paramètre	Type	Description
`SkipHostnameVerification` Par défaut : false	bool	Ignore la vérification du nom d'hôte des connexions TLS. Ce paramètre ne concerne que les clusters chiffrés. Lorsqu'il est défini sur True, il désactive la vérification du nom d'hôte. La désactivation de la vérification signifie que vous ne pouvez pas authentifier l'identité du cluster auquel vous vous connectez, ce qui pose des risques de sécurité. Par défaut, la vérification du nom d'hôte est activée.
`RouteManagerEnabled` Par défaut : false	`bool`	Cet indicateur est utilisé pour supprimer les routes confrontées à des erreurs réseau.
`RequestTimeout` par défaut : 60 * Time.second	`time.Duration`	Cela définit le temps maximum pendant lequel le client attend une réponse de DAX. Priorité : délai d'expiration du contexte (s'il est défini) > `RequestTimmeout` (s'il est défini) > `RequestTimeout` de 60 s par défaut.
`WriteRetries` Par défaut : 2	`number`	Nombre de nouvelles tentatives pour des demandes d'écriture qui ont échoué.
`ReadRetries` Par défaut : 2	`number`	Nombre de nouvelles tentatives pour des demandes de lecture qui ont échoué.

Paramètre	Type	Description
`RetryDelay` Par défaut : 0	`time.Duration`	Délai en cas d'erreurs non limitées (en secondes) pour les nouvelles tentatives en cas d'échec d'une demande.
`Logger` facultatif	`logging.Logger`	Logger est une interface permettant de journaliser les entrées dans certaines classifications.
`LogLevel` utilitaires par défaut. LogOff	`number`	Ce niveau de journalisation est défini pour DAX uniquement. Il peut être importé à l'aide de github. com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )`

> **Note**
>
> Pour `time.Duration`, l'unité par défaut est la nanoseconde. Si nous ne spécifions aucune unité pour aucun paramètre, cela sera considéré comme des nanosecondes : `daxCfg.ClusterUpdateInterval = 10` signifie 10 nanosecondes. (`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` signifie 10 millisecondes).

Création de client

Pour créer un client DAX :

- Créez une configuration DAX, puis créez un client DAX à l'aide de la configuration DAX. Vous pouvez ainsi remplacer une configuration DAX si nécessaire.

```
import (
"github.com/aws/aws-dax-go-v2/dax/utils"
"github.com/aws/aws-dax-go-v2/dax"
)

// Non - Encrypted : 'dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.
// Encrypted : daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.

config := dax.DefaultConfig()
config.HostPorts = []string{endpoint}
config.Region = region
config.LogLevel = utils.LogDebug
daxClient, err := dax.New(config)
```

Migration vers le kit SDK Go DAX V2

Ce guide de migration vous aidera à effectuer la transition de vos applications Go DAX existantes.

Utilisation du kit SDK Go DAX V1

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-dax-go/dax"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
)

func main() {
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
```

```
    // Create session
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String(region),
    })
    if err != nil {
        fmt.Printf("Failed to create session: %v\n", err)
        os.Exit(1)
    }

    // Configure DAX client
    cfg := dax.DefaultConfig()
    cfg.HostPorts = []string{endpoint}
    cfg.Region = region

    // Create DAX client
    daxClient, err := dax.New(cfg)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]*dynamodb.AttributeValue{
            "pk": {
                N: aws.String("1"),
            },
            "sk": {
                N: aws.String("1"),
            },
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
```

```
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Utilisation du kit SDK Go DAX V2

```
package main

import (
    "context"
    "fmt"
    "os"

    "github.com/aws/aws-dax-go-v2/dax"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/aws"
)

func main() {
    ctx := context.Background()
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create DAX config
    config := dax.DefaultConfig()
    // Specify Endpoint and Region
    config.HostPorts = []string{endpoint}
    config.Region = region
    // Enabling logging
    config.LogLevel = utils.LogDebug
    // Create DAX client
    daxClient, err := dax.New(config)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create key using attributevalue.Marshal for type safety
    pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
    if err != nil {
```

```
        return fmt.Errorf("error marshaling pk: %v", err)
    }
    sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
    if err != nil {
        return fmt.Errorf("error marshaling sk: %v", err)
    }

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]types.AttributeValue{
             "pk": pk,
             "sk": sk,
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(ctx, input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Pour plus d'informations sur l'utilisation de l'API, consultez les exemples AWS.

Java et DAX

Le kit SDK DAX pour Java 2.x est compatible avec le Kit SDK AWS for Java 2.x. Il repose sur Java 8+ et inclut la prise en charge des E/S non bloquantes. Pour plus d'informations sur l'utilisation de DAX avec le SDK pour AWS Java 1.x, consultez. Utilisation de DAX avec le kit SDK AWS pour Java 1.x

Utilisation du client en tant que dépendance Maven

Suivez la procédure pour utiliser le client pour le kit SDK DAX pour Java dans votre application comme dépendance.

1. Téléchargez et installez Apache Maven. Pour plus d'informations, consultez Downloading Apache Maven et Installing Apache Maven.

2. Ajoutez la dépendance de client Maven au fichier POM (Project Object Model) de votre application. Dans cet exemple, remplacez *x.x.x* par le numéro de version réel du client.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

TryDax exemple de code

Après avoir configuré votre espace de travail et ajouté le kit SDK DAX en tant que dépendance, copiez TryDax.java dans votre projet.

Exécutez le code à l'aide de cette commande.

```
java -cp classpath TryDax
```

Vous devez visualiser des résultats similaires à ce qui suit.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
  Total time: 4390.240 ms - Avg time: 4.390 ms
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
```

```
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem` et `Query`. Dans ce cas, vous avez exécuté le programme par rapport au point de terminaison DynamoDB. Vous allez maintenant réexécuter le programme mais, cette fois, par rapport à votre cluster DAX.

Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

- Sur la console DynamoDB, sélectionnez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- À l'aide du AWS CLI, entrez la commande suivante :

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  L'adresse, le port et l'URL du point de terminaison de cluster apparaissent dans la sortie, comme dans l'exemple suivant.

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

Maintenant, réexécutez le programme mais, cette fois, spécifiez l'URL du point de terminaison du cluster en tant que paramètre de ligne de commande.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Examinez la sortie et notez les informations de durée. Les durées d'exécution des opérations `GetItem` et `Query` devraient être sensiblement plus courtes avec DAX qu'avec DynamoDB.

Métriques SDK

Avec le SDK DAX pour Java 2.x, vous pouvez collecter des métriques sur les clients du service dans votre application et analyser les résultats sur Amazon. CloudWatch Pour plus d'informations, consultez Activer les métriques du kit SDK.

> **Note**
>
> Le kit SDK DAX pour Java collecte uniquement les métriques `ApiCallSuccessful` et `ApiCallDuration`.

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();
```

```
        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
```

```
                .tableName(tableName)
                .keySchema(KeySchemaElement.builder()
                        .keyType(KeyType.HASH)
                        .attributeName("pk")
                        .build(), KeySchemaElement.builder()
                        .keyType(KeyType.RANGE)
                        .attributeName("sk")
                        .build())
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName("pk")
                        .attributeType(ScalarAttributeType.N)
                        .build(), AttributeDefinition.builder()
                        .attributeName("sk")
                        .attributeType(ScalarAttributeType.N)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }
```

```
   private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
       System.out.println("Writing data to the table...");

       int stringSize = 1000;
       StringBuilder sb = new StringBuilder(stringSize);
       for (int i = 0; i < stringSize; i++) {
           sb.append('X');
       }
       String someData = sb.toString();

       try {
           for (int ipk = 1; ipk <= pkmax; ipk++) {
               System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
               for (int isk = 1; isk <= skmax; isk++) {
                   client.putItem(PutItemRequest.builder()
                           .tableName(tableName)
                           .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                           .build()).get();
               }
           }
       } catch (Exception e) {
           System.err.println("Unable to write item:");
           e.printStackTrace();
       }
   }

   private static AttributeValue attr(int n) {
       return AttributeValue.builder().n(String.valueOf(n)).build();
   }

   private static AttributeValue attr(String s) {
       return AttributeValue.builder().s(s).build();
   }

   private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
       long startTime, endTime;
       System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);
```

```
        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (int ipk = 1; ipk <= pk; ipk++) {
                    for (int isk = 1; isk <= sk; isk++) {
                        client.getItem(GetItemRequest.builder()
                                .tableName(tableName)
                                .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                                .build()).get();
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            for (int ipk = 1; ipk <= pk; ipk++) {
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
                    e.printStackTrace();
                }
            }
```

```
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

.NET et DAX

Suivez ces étapes pour exécuter l'exemple .NET sur votre EC2 instance Amazon.

> **Note**
>
> Ce didacticiel utilise le SDK .NET 9. Il montre comment exécuter un programme dans votre VPC Amazon par défaut pour accéder à votre cluster Amazon DynamoDB Accelerator (DAX). Il fonctionne avec le AWS SDK v4 pour .NET. Pour plus de détails sur les modifications apportées à la version 4 et des informations sur la migration, consultez la section Migration vers la version 4 du AWS SDK pour .NET. Si vous préférez, vous pouvez utiliser le AWS Toolkit for Visual Studio pour écrire une application .NET et la déployer dans votre VPC. Pour plus d'informations, consultez Création et déploiement d'applications Elastic Beanstalk dans .NET à l'aide d'AWS Toolkit for Visual Studio dans le Guide du développeur AWS Elastic Beanstalk.

Pour exécuter l'exemple .NET pour DAX

1. Accédez à la page des téléchargements Microsoft et téléchargez le dernier SDK .NET 9 pour Linux. Le fichier téléchargé est `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Extrayez les fichiers du kit SDK.

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   Remplacez *N.N.N* par le numéro de version réel du kit SDK .NET, par exemple : `9.0.305`.

3. Vérifiez l'installation.

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

Le numéro de version du kit SDK .NET doit alors s'imprimer.

> **Note**
>
> Au lieu du numéro de version, vous pouvez recevoir l'erreur suivante :
> erreur : libunwind.so.8 : impossible d'ouvrir le fichier objet partagé: Fichier ou répertoire inexistant
> Pour résoudre l'erreur, installez le package `libunwind`.
>
> ```
> sudo yum install -y libunwind
> ```
>
> Une fois que vous avez terminé, vous devez pouvoir exécuter la commande `dotnet --version` sans erreur.

4. Crée un projet .NET.

```
dotnet new console -o myApp
```

Cela nécessite quelques minutes pour effectuer une one-time-only configuration. Une fois l'opération terminée, exécutez l'exemple de projet.

```
dotnet run --project myApp
```

Vous devriez recevoir le message suivant : `Hello World!`

5. Le fichier `myApp/myApp.csproj` contient les métadonnées de votre projet. Pour utiliser le client DAX dans votre application, modifiez le fichier de telle sorte qu'il ressemble à ceci.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
```

```
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Téléchargez le code source de l'exemple de programme (fichier `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Une fois le téléchargement terminé, extrayez les fichiers source.

```
unzip TryDax.zip
```

7. Exécutez maintenant les exemples de programmes de DotNet, un par un. Pour chaque programme, copiez son contenu dans le fichier `myApp/Program.cs`, puis exécutez le projet `MyApp`.

Exécutez les programmes .NET suivants. Le premier programme crée une table DynamoDB nommée `TryDaxTable`. Le deuxième programme écrit des données dans la table.

```
cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
dotnet run --project myApp

cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
dotnet run --project myApp
```

8. Ensuite, exécutez certains programmes pour effectuer les opérations `GetItem`, `Query` et `Scan` sur votre cluster DAX. Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

- Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- À l'aide de AWS CLI— Entrez la commande suivante.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

À présent, exécutez les programmes suivants, en spécifiant le point de terminaison de votre cluster comme paramètre de ligne de commande. (Remplacez l'exemple de point de terminaison par le point de terminaison réel de votre cluster DAX.)

```
cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

9. Exécutez le programme .NET suivant pour supprimer `TryDaxTable`.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Pour plus d'informations sur ces programmes, consultez les sections suivantes :

- 0-1 CreateTable pièce
- 02-Write-Data.cs
- 03- GetItem -Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs

- DeleteTable0,6 à 6 pièces

0-1 CreateTable pièce

Le programme `01-CreateTable.cs` crée une table (`TryDaxTable`). Les programmes .NET restants de cette section dépendent de cette table.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };
```

```
            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

02-Write-Data.cs

Le programme `02-Write-Data.cs` écrit des données de test dans `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
```

```
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                        }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

03- GetItem -Test.cs

Le programme `03-GetItem-Test.cs` exécute des opérations `GetItem` sur `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);
```

```
            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

Le programme `04-Query-Test.cs` exécute des opérations `Query` sur `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");


            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
 and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
```

```
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

05-Scan-Test.cs

Le programme `05-Scan-Test.cs` exécute des opérations `Scan` sur `TryDaxTable`.

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
```

```
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

DeleteTable0,6 à 6 pièces

Le programme `06-DeleteTable.cs` supprime `TryDaxTable`. Exécutez ce programme après avoir fini les tests.

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
```

```
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

Python et DAX

Suivez cette procédure pour exécuter l'exemple d'application Python sur votre EC2 instance Amazon.

Pour exécuter l’exemple Python pour DAX

1. Installez le client Python DAX à l’aide de l’utilitaire `pip`.

   ```
   pip install amazon-dax-client
   ```

2. Téléchargez le code source de l’exemple de programme (fichier `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Une fois le téléchargement terminé, extrayez les fichiers source.

   ```
   unzip TryDax.zip
   ```

3. Exécutez les programmes Python suivants. Le premier programme crée une table Amazon DynamoDB nommée `TryDaxTable`. Le deuxième programme écrit des données dans la table.

```
python 01-create-table.py
python 02-write-data.py
```

4. Exécutez les programmes Python suivants.

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

   Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

5. A l'étape précédente, vous avez exécuté les programmes par rapport au point de terminaison DynamoDB. A présent, réexécutez-les mais, cette fois, les opérations `GetItem`, `Query` et `Scan` sont traitées par votre cluster DAX.

   Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

   - Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - À l'aide de AWS CLI— Entrez la commande suivante.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Le point de terminaison du cluster apparaît dans la sortie, comme dans cet exemple.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

À présent, réexécutez les programmes, mais cette fois, spécifiez le point de terminaison du cluster en tant que paramètre de ligne de commande.

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observez le reste de la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX par rapport à DynamoDB.

6. Exécutez le programme Python suivant pour supprimer `TryDaxTable`.

```
python 06-delete-table.py
```

Pour plus d'informations sur ces programmes, consultez les sections suivantes :

- 01-create-table.py
- 02-write-data.py
- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

01-create-table.py

Le programme `01-create-table.py` crée une table (`TryDaxTable`). Les programmes Python restants de cette section dépendent de cette table.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.
```

```
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

Le programme `02-write-data.py` écrit des données de test dans `TryDaxTable`.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
```

```
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

Le programme `03-getitem-test.py` exécute des opérations `GetItem` sur `TryDaxTable`. Cet exemple est donné pour la Région eu-west-1.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
```

```
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
                })
                print('.', end='')
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
```

```
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

04-query-test.py

Le programme `04-query-test.py` exécute des opérations `Query` sur `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)
```

```
    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

Le programme `05-scan-test.py` exécute des opérations `Scan` sur `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()
```

```
    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

Le programme `06-delete-table.py` supprime `TryDaxTable`. Exécutez ce programme après avoir fini de tester la fonctionnalité d'Amazon DynamoDB Accelerator (DAX).

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

Modification d’une application existante pour utiliser DAX

Si vous disposez déjà d’une application Java utilisant Amazon DynamoDB, vous pouvez la modifier pour qu’elle puisse accéder à votre cluster DynamoDB Accelerator (DAX). Il n'est pas nécessaire de réécrire l'application dans son intégralité car le client Java DAX est similaire au client de bas niveau DynamoDB inclus dans le SDK pour Java 2.x.AWS Pour plus de détails, consultez Utilisation d’éléments dans DynamoDB.

> **Note**
>
> Cet exemple utilise le AWS SDK pour Java 2.x. Pour le kit SDK pour Java 1.x hérité, consultez Modification d’une application du kit SDK pour Java 1.x pour utiliser DAX.

Pour modifier votre programme, remplacez le client DynamoDB par un client DAX.

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

Vous pouvez également utiliser la bibliothèque de haut niveau qui fait partie du AWS SDK pour Java 2.x, en remplaçant le client DynamoDB par un client DAX.

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
```

```
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.builder()
        .dynamoDbClient(dax)
        .build();
```

Pour plus d'informations, consultez Mappage d'éléments dans des tables DynamoDB.

Gestion des clusters DAX

Cette section traite de certaines des tâches de gestion courantes pour les clusters Amazon DynamoDB Accelerator (DAX).

Rubriques

- Autorisations IAM pour gérer un cluster DAX
- Mise à l'échelle d'un cluster DAX
- Personnalisation des paramètres de cluster DAX
- Configuration des paramètres de durée de vie (TTL)
- Prise en charge de l'étiquetage pour DAX
- AWS CloudTrail intégration
- Suppression d'un cluster DAX

Autorisations IAM pour gérer un cluster DAX

Lorsque vous administrez un cluster DAX à l'aide du AWS Management Console ou du AWS Command Line Interface(AWS CLI), nous vous recommandons vivement de limiter le champ des actions que les utilisateurs peuvent effectuer. Vous limiterez ainsi les risques tout en suivant le principe du moindre privilège.

La discussion suivante porte sur le contrôle d'accès pour la gestion APIs du DAX. Pour plus d'informations, consultez Amazon DynamoDB accelerator dans la Référence d'API Amazon DynamoDB.

> **Note**
>
> Pour des informations plus détaillées sur la gestion des autorisations Gestion des identités et des accès AWS(IAM), consultez les rubriques suivantes :
>
> - IAM et création de clusters DAX : Création d'un cluster DAX.
> - Opérations de plan de données IAM et DAX : Contrôle d'accès à DAX.

Pour la gestion du DAX APIs, vous ne pouvez pas étendre les actions d'API à une ressource spécifique. L'élément `Resource` doit être défini sur `"*"`. Cela diffère des opérations d'API de plan de données DAX, telles que `GetItem`, `Query` et `Scan`. Les opérations de plan de données sont exposées via le client DAX, et ces opérations peuvent être limitées à des ressources spécifiques.

A titre d'illustration, prenez en compte le document de politique IAM suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Supposons que le but de cette politique soit d'autoriser les appels d'API de gestion DAX pour le cluster `DAXCluster01`, et uniquement pour ce cluster.

Supposons maintenant qu'un utilisateur émette la AWS CLI commande suivante.

```
aws dax describe-clusters
```

Cette commande échouera avec une exception Not Authorized (Non autorisé), car l'appel d'API `DescribeClusters` sous-jacent ne peut pas être attribué à un cluster spécifique. Bien que la stratégie soit valide au niveau de la syntaxe, la commande échoue car l'élément `Resource` doit être défini sur `"*"`. Cependant, si l'utilisateur exécute un programme qui envoie des appels de plan de données DAX (tels que `GetItem` ou `Query`) à `DAXCluster01`, ces appels aboutissent. Cela est dû au fait que le plan de données DAX APIs peut être étendu à des ressources spécifiques (dans ce cas,`DAXCluster01`).

Si vous souhaitez rédiger une seule stratégie IAM complète englobant à la fois la gestion DAX APIs et le plan de données DAX APIs, nous vous suggérons d'inclure deux déclarations distinctes dans le document de stratégie. L'une de ces instructions doit porter sur le plan de données DAX APIs, tandis que l'autre concerne la gestion du DAX. APIs

L'exemple de stratégie ci-dessous montre cette approche. Notez que l'instruction `DAXDataAPIs` est attribuée à la ressource `DAXCluster01`, mais la ressource pour `DAXManagementAPIs` doit être `"*"`. Les actions affichées dans chaque instruction ne le sont qu'à titre d'illustration. Vous pouvez personnaliser ces ressources en fonction des besoins de votre application.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
```

```
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Mise à l'échelle d'un cluster DAX

Les options disponibles pour la mise à l'échelle d'un cluster DAX sont au nombre de deux. La première option correspond à la mise à l'échelle horizontale, qui consiste à ajouter des réplicas en lecture au cluster. La deuxième option correspond à la mise à l'échelle verticale, où vous sélectionnez différents types de nœuds. Pour obtenir des conseils sur la façon de choisir une taille de cluster et un type de nœud appropriés pour votre application, veuillez consulter Guide de dimensionnement de cluster DAX.

Mise à l'échelle horizontale

Avec la mise à l'échelle horizontale, vous pouvez améliorer le débit des opérations de lecture en ajoutant davantage de réplicas en lecture au cluster. Un même cluster DAX prend en charge jusqu'à 10 réplicas en lecture, et vous pouvez ajouter ou supprimer des réplicas pendant que le cluster s'exécute.

Lorsque vous ajoutez un nouveau nœud, vous devez synchroniser les données du cache à partir d'un nœud pair. Par conséquent, le délai supplémentaire varie en fonction de la taille du cache et de la charge de travail de votre application. Comme bonne pratique, nous vous recommandons de préalablement mettre à l'échelle votre cluster pour répondre aux pics de trafic attendus. Pour plus d'informations sur les directives de mise à l'échelle correcte et les recommandations de surveillance, consultez Guide de dimensionnement de cluster DAX.

Les AWS CLI exemples suivants montrent comment augmenter ou diminuer le nombre de nœuds. L'argument `--new-replication-factor` spécifie le nombre total de nœuds du cluster. L'un des nœuds est le nœud principal et les autres nœuds sont des réplicas en lecture.

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> **Note**
>
> Le statut du cluster passe à `modifying` lorsque vous modifiez le facteur de réplication. Le statut devient `available` lorsque la modification est terminée.

Mise à l'échelle verticale

Si vous disposez d'un jeu de données actif volumineux, l'utilisation de types de nœuds plus grands peut s'avérer bénéfique pour votre application. En effet, les nœuds de grande taille peuvent conférer au cluster une plus grande capacité de stockage de données en mémoire, ce qui limite les échecs de cache et améliore les performances globales de l'application. (Tous les nœuds dans un cluster DAX doivent être du même type.)

Si votre cluster DAX présente un taux élevé d'échecs d'opérations d'écriture ou de cache, il se peut que votre application bénéficie également de l'utilisation de types de nœuds plus importants. Les opérations d'écriture et les échecs de cache utilisent des ressources sur le nœud principal du cluster. Par conséquent, l'utilisation de types de nœuds plus importants peut augmenter les performances du nœud principal et permettre ainsi un débit plus élevé pour ces types d'opérations.

Vous ne pouvez pas modifier les types de nœuds sur un cluster DAX en cours d'exécution. Au lieu de cela, vous devez créer un nouveau cluster avec le type de nœud souhaité. Pour obtenir la liste des types de nœuds pris en charge, consultez Nœuds.

Vous pouvez créer un nouveau cluster DAX à l'aide du AWS Management ConsoleCloudFormationAWS CLI, du ou du AWS SDK. (Pour le AWS CLI, utilisez le `--node-type` paramètre pour spécifier le type de nœud.)

Personnalisation des paramètres de cluster DAX

Lors de la création d'un cluster DAX, les paramètres par défaut utilisés sont les suivants :

- Expulsion de cache automatique activée avec un time-to-live (TTL) de 5 minutes
- Aucune préférence pour les zones de disponibilité
- Aucune préférence pour les fenêtres de maintenance
- Notifications désactivées

Pour les nouveaux clusters, vous pouvez personnaliser les paramètres au moment de leur création. Pour ce faire, dans la AWS Management Console, désélectionnez Utiliser les paramètres par défaut pour modifier les paramètres suivants :

- Réseau et sécurité : vous permet d'exécuter des nœuds de cluster DAX individuels dans différentes zones de disponibilité de la région actuelle AWS. Si vous choisissez Aucune préférence, les nœuds sont répartis automatiquement entre les zones de disponibilité.
- Parameter Group (Groupe de paramètres) – Ensemble nommé de paramètres appliqués à chaque nœud du cluster. Vous pouvez utiliser un groupe de paramètres pour spécifier le comportement du paramètre de durée de vie (TTL) du cache. Vous pouvez modifier la valeur d'un paramètre donné dans un groupe de paramètres à tout moment (à l'exception de `default.dax.1.0` du groupe de paramètres par défaut).
- Maintenance Window (Fenêtre de maintenance) – Temps hebdomadaire pendant lequel les mises à niveau et les correctifs logiciels sont appliqués aux nœuds du cluster. Vous pouvez choisir le

jour de début, l'heure de début et la durée de la fenêtre de maintenance. Si vous choisissez No Preference (Aucune préférence), la fenêtre de maintenance est sélectionnée de façon aléatoire dans une plage de temps de 8 heures par région. Pour de plus amples informations, veuillez consulter Fenêtre de maintenance.

> **Note**
>
> Le groupe de paramètres et la fenêtre de maintenance peuvent également être modifiés à tout moment sur un cluster en cours d'exécution.

Quand un événement de maintenance se produit, DAX peut vous avertir à l'aide d'Amazon Simple Notification Service (Amazon SNS). Pour configurer les notifications, choisissez une option dans le sélecteur Rubrique pour notification SNS. Vous pouvez créer une rubrique Amazon SNS ou utiliser une rubrique existante.

Pour plus d'informations sur la configuration d'une rubrique Amazon SNS et sur l'abonnement à celle-ci, consultez Mise en route avec Amazon SNS dans le Guide du développeur Amazon Simple Notification Service.

Configuration des paramètres de durée de vie (TTL)

DAX gère deux caches pour les données lues à partir de DynamoDB :

- Cache d'élément – Pour les éléments extraits à l'aide des opérations `GetItem` ou `BatchGetItem`.
- Cache de requête – Pour les jeux de résultats extraits à l'aide des opérations `Query` ou `Scan`.

Pour plus d'informations, consultez Cache d'élément et Cache de requête.

La durée de vie (TTL) par défaut de chacun de ces caches est de 5 minutes. Si vous souhaitez utiliser d'autres paramètres TTL, vous pouvez lancer un cluster DAX en utilisant un groupe de paramètres personnalisés. Pour ce faire, dans la console, choisissez DAX | Groupes de paramètres dans le volet de navigation.

Vous pouvez également effectuer ces tâches à partir de AWS CLI. L'exemple suivant montre comment lancer un nouveau cluster DAX en utilisant un groupe de paramètres personnalisé. Dans cet exemple, la durée de vie (TTL) du cache d'élément définie est de 10 minutes et celle du cache de requête est de 3 minutes.

1. Créez un groupe de paramètres.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Définissez la durée de vie de cache de l'élément sur 10 minutes (600 000 millisecondes).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Définissez la durée de vie de cache de la requête sur 3 minutes (180 000 millisecondes).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Vérifiez que les paramètres ont été définis correctement.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

Vous pouvez maintenant lancer un nouveau cluster DAX avec ce groupe de paramètres.

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> Vous ne pouvez pas modifier un groupe de paramètres utilisé par une instance DAX en cours d'exécution.

Prise en charge de l’étiquetage pour DAX

De nombreux AWS services, dont DynamoDB, prennent en charge le balisage, c'est-à-dire la possibilité d'étiqueter les ressources avec des noms définis par l'utilisateur. Vous pouvez attribuer des balises aux clusters DAX, ce qui vous permet d'identifier rapidement toutes vos AWS ressources qui ont le même tag, ou de classer vos AWS factures en fonction des tags que vous attribuez.

Pour de plus amples informations, veuillez consulter Ajout de balises et d’étiquettes aux ressources dans DynamoDB.

À l'aide du AWS Management Console

Pour gérer des étiquettes de cluster DAX

1. Ouvrez la console DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation, sous DAX, choisissez Clusters.
3. Choisissez le cluster que vous souhaitez utiliser.
4. Sélectionnez l’onglet Tags (Identifications). Vous pouvez ajouter, répertorier, modifier ou supprimer vos balises ici.

   Lorsque les paramètres vous conviennent, choisissez Apply Changes.

À l'aide du AWS CLI

Lorsque vous utilisez les balises de cluster AWS CLI pour gérer le DAX, vous devez d'abord déterminer le nom de ressource Amazon (ARN) du cluster. L’exemple suivant montre comment déterminer l’ARN pour un cluster nommé `MyDAXCluster`.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

Dans la sortie, l’ARN ressemblera à ceci : `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

L’exemple suivant montre comment baliser le cluster.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
```

```
    --tags="Key=ClusterUsage,Value=prod"
```

Affiche toutes les balises d'un cluster.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Pour supprimer une balise, vous devez spécifier sa clé.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail intégration

DAX est intégréAWS CloudTrail, ce qui vous permet d'auditer les activités du cluster DAX. Vous pouvez utiliser les CloudTrail journaux pour afficher toutes les modifications apportées au niveau du cluster. Vous pouvez aussi afficher les modifications apportées aux composants d'un cluster, tels que nœuds, groupes de sous-réseau et groupes de paramètres. Pour de plus amples informations, veuillez consulter Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail.

Suppression d'un cluster DAX

Si vous n'utilisez plus un cluster DAX, supprimez-le afin d'éviter d'être facturé pour des ressources non utilisées.

Vous pouvez supprimer un cluster DAX à l'aide de la console ou de l'AWS CLI. Voici un exemple.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Surveillance de l'accélérateur DynamoDB

La surveillance joue un rôle important dans le maintien de la fiabilité, de la disponibilité et des performances d'Amazon DynamoDB Accelerator (DAX) et de vos solutions.AWS Vous devez collecter des données de surveillance provenant de toutes les parties de votre AWS solution afin de pouvoir corriger plus facilement une défaillance multipoint, le cas échéant.

Avant de commencer la surveillance de DAX, vous devez créer un plan de surveillance contenant les réponses aux questions suivantes :

- Quels sont les objectifs de la surveillance ?
- Quelles sont les ressources à surveiller ?
- À quelle fréquence les ressources doivent-elles être surveillées ?
- Quels outils de surveillance utiliser ?
- Qui exécute les tâches de supervision ?
- Qui doit être informé en cas de problème ?

Rubriques

- Outils de surveillance pour DynamoDB Accelerator
- Surveillance avec Amazon CloudWatch
- Journalisation des opérations DAX avec AWS CloudTrail

Outils de surveillance pour DynamoDB Accelerator

AWS fournit des outils que vous pouvez utiliser pour surveiller Amazon DynamoDB Accelerator (DAX). Vous pouvez configurer certains outils pour qu'ils effectuent la surveillance automatiquement ; d'autres nécessitent une intervention manuelle. Nous vous recommandons d'automatiser le plus possible les tâches de supervision.

Rubriques

- Outils de surveillance automatique
- Outils de surveillance manuelle

Outils de surveillance automatique

Pour surveiller DAX et signaler d'éventuels problème, vous pouvez utiliser les outils de surveillance automatique suivants :

- Amazon CloudWatch Alarms : surveillez une seule métrique sur une période que vous spécifiez et effectuez une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné sur un certain nombre de périodes. L'action est une notification envoyée à une rubrique Amazon Simple Notification Service (Amazon SNS) ou à une politique Amazon EC2 Auto Scaling. CloudWatch les alarmes n'appellent pas d'actions simplement parce qu'elles sont dans un état particulier ; l'état doit avoir changé et être maintenu pendant un certain nombre de périodes. Pour

de plus amples informations, veuillez consulter Surveillance des métriques dabs DynamoDB avec Amazon CloudWatch.

- Amazon CloudWatch Logs — Surveillez, stockez et accédez à vos fichiers journaux depuis AWS CloudTrail ou d'autres sources. Pour plus d'informations, consultez la section Monitoring Log Files dans le guide de CloudWatch l'utilisateur Amazon.
- Amazon CloudWatch Events : associez les événements et acheminez-les vers une ou plusieurs fonctions ou flux cibles afin d'apporter des modifications, de recueillir des informations d'état et de prendre des mesures correctives. Pour plus d'informations, consultez la section Qu'est-ce qu'Amazon CloudWatch Events dans le guide de CloudWatch l'utilisateur Amazon.
- AWS CloudTrail Surveillance des journaux : partagez les fichiers journaux entre les comptes, surveillez les fichiers CloudTrail CloudWatch journaux en temps réel en les envoyant à Logs, écrivez des applications de traitement des journaux en Java et vérifiez que vos fichiers journaux n'ont pas changé après leur livraison par CloudTrail. Pour plus d'informations, consultez la section Utilisation des fichiers CloudTrail journaux dans le guide de AWS CloudTrail l'utilisateur.

Outils de surveillance manuelle

Un autre élément important de la surveillance du DAX consiste à surveiller manuellement les éléments non couverts par les CloudWatch alarmes. Les AWS Management Console tableaux de bord DAX CloudWatch Trusted Advisor,, et autres fournissent une at-a-glance vue d'ensemble de l'état de votre AWS environnement. Nous recommandons de consulter également les fichiers journaux sur DAX.

- Le tableau de bord DAX présente les éléments suivants :
  - État du service
- La page CloudWatch d'accueil affiche les informations suivantes :
  - Alarmes et statuts en cours
  - Graphiques des alarmes et des ressources
  - Statut d’intégrité du service

  En outre, vous pouvez utiliser CloudWatch pour effectuer les opérations suivantes :
  - Créer des tableaux de bord personnalisés pour surveiller les services qui vous intéressent.
  - Données de métriques de graphiques pour résoudre les problèmes et découvrir les tendances.
  - Recherchez et parcourez tous les indicateurs de vos AWS ressources.
  - Créer et Modifier des alarmes pour être informé des problèmes.

Surveillance avec Amazon CloudWatch

Vous pouvez surveiller DynamoDB Accelerator (DAX) à l'aide d' CloudWatchAmazon, qui collecte et traite les données brutes du DAX pour en faire des métriques lisibles en temps quasi réel. Ces statistiques sont enregistrées pour une durée de deux semaines. Vous pouvez ensuite accéder aux informations historiques pour acquérir un meilleur point de vue de la façon dont votre service ou application web s'exécute. Par défaut, les données métriques DAX sont envoyées CloudWatch automatiquement à. Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch ? dans le guide de CloudWatch l'utilisateur Amazon.

Rubriques

- Comment utiliser les métriques DAX ?
- Affichage des métriques et dimensions DAX
- Création d' CloudWatch alarmes pour surveiller le DAX
- Surveillance en production

Comment utiliser les métriques DAX ?

Les métriques présentées par DAX fournissent des informations que vous pouvez analyser de diverses manières. La liste suivante présente certaines utilisations courantes des métriques. Il s'agit de quelques suggestions pour vous aider à démarrer et non d'une liste exhaustive.

Comment puis-je ?	Métriques pertinentes
Déterminer si des erreurs système se sont produites	Surveillez `FaultRequestCount` pour déterminer si des demandes ont entraîné un code HTTP 500 (erreur serveur). Cela peut indiquer une erreur de service interne DAX ou un HTTP 500 dans la SystemErrors métrique de la table sous-jacente.
Déterminer si des erreurs utilisateur se sont produites	Surveillez `ErrorRequestCount` pour déterminer si des demandes ont entraîné un code HTTP 400 (erreur client). Si vous constatez une augmentation du nombre d'erreurs, vous pouvez en rechercher l'origine afin de vous assurer que vous envoyez des demandes client correctes.

Comment puis-je ?	Métriques pertinentes
Déterminer si des échecs de cache se sont produits	Surveillez `ItemCacheMisses` pour déterminer le nombre de fois où un élément n'a pas été trouvé dans le cache, ainsi que `QueryCacheMisses` et `ScanCacheMisses` pour déterminer le nombre de fois où un résultat de requête ou d'analyse n'a pas été trouvé dans le cache.
Surveiller le taux d'accès au cache	Utilisez CloudWatch Metric Math pour définir une métrique du taux de réussite du cache à l'aide d'expressions mathématiques. Par exemple, pour le cache d'élément, vous pouvez utiliser l'expression m1/SUM([m1, m2])*100, où m1 est la métrique `ItemCacheHits` et m2, la métrique `ItemCacheMisses` pour votre cluster. Pour les caches de requête et d'analyse, vous pouvez suivre le même schéma en utilisant la métrique de cache de requête et d'analyse correspondante.

Affichage des métriques et dimensions DAX

Lorsque vous interagissez avec Amazon DynamoDB, celui-ci envoie des métriques et des dimensions à Amazon. CloudWatch Vous pouvez utiliser les procédures suivantes pour afficher les métriques pour DynamoDB Accelerator (DAX).

Pour consulter les métriques (console)

Les métriques sont d'abord regroupées par espace de noms de service, puis par les différentes combinaisons de dimension au sein de chaque espace de noms.

1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Dans le panneau de navigation, sélectionnez Métriques.
3. Sélectionnez l'espace de noms DAX.

Pour afficher les métriques (AWS CLI)

- À partir d'une invite de commande, utilisez la commande suivante :

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Métriques et dimensions DAX

Les sections suivantes contiennent les métriques et les dimensions auxquelles DAX envoie des données. CloudWatch

Métriques DAX

Les métriques suivantes sont disponibles dans DAX. DAX envoie des métriques CloudWatch uniquement lorsqu'elles ont une valeur différente de zéro.

> **Note**
>
> CloudWatch agrège les métriques DAX suivantes à des intervalles d'une minute :
>
> - `CPUUtilization`
> - `CacheMemoryUtilization`
> - `NetworkBytesIn`
> - `NetworkBytesOut`
> - `BaselineNetworkBytesInUtilization`
> - `BaselineNetworkBytesOutUtilization`
> - `NetworkPacketsIn`
> - `NetworkPacketsOut`
> - `GetItemRequestCount`
> - `BatchGetItemRequestCount`
> - `BatchWriteItemRequestCount`
> - `DeleteItemRequestCount`
> - `PutItemRequestCount`
> - `UpdateItemRequestCount`
> - `TransactWriteItemsCount`
> - `TransactGetItemsCount`
> - `ItemCacheHits`
> - `ItemCacheMisses`

- QueryCacheHits
- QueryCacheMisses
- ScanCacheHits
- ScanCacheMisses
- TotalRequestCount
- ErrorRequestCount
- FaultRequestCount
- FailedRequestCount
- QueryRequestCount
- ScanRequestCount
- ClientConnections
- EstimatedDbSize
- EvictedSize
- CPUCreditUsage
- CPUCreditBalance
- CPUSurplusCreditBalance
- CPUSurplusCreditsCharged

Certaines statistiques, telles que `Average` ou `Sum`, s'appliquent à chaque métrique. Cependant, toutes ces valeurs sont disponibles via la console DAX, ou à l'aide de la CloudWatch console AWS CLI, ou AWS SDKs pour toutes les métriques. Dans le tableau suivant, chaque métrique dispose d'une liste de statistiques valides applicables à cette métrique.

Métrique	Description
`CPUUtilization`	Pourcentage d'utilisation de l'UC du nœud ou du cluster. Unités `Percent`: Statistiques valides : • `Minimum` • `Maximum`

Métrique	Description
	• Average
CacheMemoryUtilization	Pourcentage de mémoire cache disponible utilisé par le cache d'éléments et le cache de requêtes sur le nœud ou le cluster. Les données mises en cache commencent à être expulsées avant que l'utilisation de la mémoire atteigne 100 % (voir métriques EvictedSize). Si la valeur CacheMemoryUtilization atteint 100 % sur un nœud quelconque, les demandes d'écriture sont limitées et vous devriez envisager de passer à un cluster dont le type de nœud est plus grand. Unités Percent: Statistiques valides : • Minimum • Maximum • Average
NetworkBytesIn	Nombre d'octets reçus par le nœud ou le cluster sur toutes les interfaces réseau. Unités Bytes: Statistiques valides : • Minimum • Maximum • Average

Métrique	Description
NetworkBytesOut	Nombre d'octets envoyés par le nœud ou le cluster sur toutes les interfaces réseau. Cette métrique identifie le volume de trafic sortant en termes de nombre d'octets sur un seul nœud ou cluster. Unités Bytes: Statistiques valides : • Minimum • Maximum • Average
BaselineNetworkBytesInUtilization	Pourcentage de bande passante du réseau de base utilisée à un moment donné pour le trafic entrant. À titre de référence , 50 % signifie que la moitié de la bande passante du réseau disponible pour le trafic entrant est utilisée. Unités Percent: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
`BaselineNetworkBytesOutUtilization`	Pourcentage de la bande passante du réseau de référence consommée à un moment donné pour le trafic sortant. À titre de référence, 50 % signifie que la moitié de la bande passante du réseau disponible pour le trafic sortant est utilisée. Unités `Percent`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`NetworkPacketsIn`	Nombre de paquets reçus par le nœud ou le cluster sur toutes les interfaces réseau. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average`

Métrique	Description
NetworkPacketsOut	Nombre de paquets envoyés par le nœud ou le cluster sur toutes les interfaces réseau. Cette métrique identifie le volume de trafic sortant en termes de nombre de paquets sur un seul nœud ou cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average
GetItemRequestCount	Nombre de demandes GetItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
BatchGetItemRequestCount	Nombre de demandes BatchGetItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum
BatchWriteItemRequestCount	Nombre de demandes BatchWriteItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
DeleteItemRequestCount	Nombre de demandes DeleteItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum
PutItemRequestCount	Nombre de demandes PutItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
UpdateItemRequestCount	Nombre de demandes UpdateItem traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum
TransactWriteItemsCount	Nombre de demandes TransactWriteItems traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
`TransactGetItemsCount`	Nombre de demandes `TransactGetItems` traitées par le nœud ou le cluster. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	Nombre de fois qu'un élément a été renvoyé du cache par le nœud ou le cluster. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`ItemCacheMisses`	Nombre de fois qu'un élément ne figurait pas dans le cache de nœuds ou de clusters, et a dû être extrait de DynamoDB. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	Nombre de fois qu'un élément a été renvoyé à partir du cache de nœuds ou de clusters. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`QueryCacheMisses`	Nombre de fois qu'un résultat de requête ne figurait pas dans le cache de nœuds ou de clusters, et a dû être extrait de DynamoDB. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanCacheHits`	Nombre de fois qu'un résultat d'analyse a été renvoyé à partir du cache de nœuds ou de clusters. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
ScanCacheMisses	Nombre de fois qu'un résultat d'analyse ne figurait pas dans le cache de nœuds ou de clusters, et a dû être extrait de DynamoDB. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum
TotalRequestCount	Nombre total de demandes traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
`ErrorRequestCount`	Nombre total de demandes ayant entraîné une erreur utilisateur signalée par le nœud ou le cluster. Les demandes qui ont été limitées par le nœud ou le cluster sont incluses. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	Nombre total de demandes limitées par le nœud ou le cluster. Les demandes qui ont été limitées par DynamoDB ne sont pas incluses et peuvent être surveillées à l'aide de métriques DynamoDB. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`FaultRequestCount`	Nombre total de demandes ayant entraîné une erreur interne signalée par le nœud ou le cluster. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Nombre total de demandes ayant entraîné une erreur signalée par le nœud ou le cluster. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
QueryRequestCount	Nombre de demandes de requête traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum
ScanRequestCount	Nombre de demandes d'analyse traitées par le nœud ou le cluster. Unités Count: Statistiques valides : • Minimum • Maximum • Average • SampleCount • Sum

Métrique	Description
`ClientConnections`	Nombre de connexions simultanées établies par des clients au nœud ou au cluster. Unités `Count`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Approximation de la quantité de données mises en cache dans les caches d'éléments et de requêtes par le nœud ou le cluster. Unités `Bytes`: Statistiques valides : • `Minimum` • `Maximum` • `Average`

Métrique	Description
`EvictedSize`	Quantité de données expulsées par le nœud ou le cluster afin de ménager de la place pour les données nouvellement demandées. Si le taux d'échecs augmente et que vous voyez cette métrique augmenter, cela signifie probablement que votre ensemble de travail a augmenté. Songez à passer à un cluster dont le type de nœud est plus grand. Unités `Bytes`: Statistiques valides : • `Minimum` • `Maximum` • `Average` • `Sum`

Métrique	Description
`CPUCreditUsage`	Nombre de crédits UC dépensés par le nœud pour l'utilisation de l'UC. Un crédit de processeur équivaut à un vCPU fonctionnant à 100 % d'utilisation pendant une minute ou une combinaison équivalente de vCPUs, d'utilisation et de temps (par exemple, un vCPU fonctionnant à 50 % d'utilisation pendant deux minutes ou deux vCPU CPUs fonctionnant à 25 % d'utilisation pendant deux minutes). Les métriques de crédits UC sont disponibles uniquement toutes les 5 minutes. Si vous spécifiez une période supérieure à cinq minutes, utilisez la statistique `Sum` au lieu de la statistique `Average`. Unités `Credits (vCPU-minutes)` : Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`CPUCreditBalance`	Nombre de crédits UC gagnés qu'un nœud a accumulés depuis son lancement ou son démarrage. Les crédits sont accumulés dans le solde de crédits quand ils sont gagnés et supprimés du solde de crédits lorsqu'ils sont dépensés. Le solde de crédits présente une limite maximum qui est déterminée par la taille de nœud DAX. Une fois que la limite est atteinte, tous les nouveaux crédits gagnés sont rejetés. Le nœud peut dépenser les crédits disponibles dans `CPUCreditBalance` pour dépasser son niveau de base d'utilisation de l'UC. Unités `Credits (vCPU-minutes)` : Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`CPUSurplusCreditBalance`	Nombre de crédits excédentaires dépensés par un nœud DAX lorsque sa valeur `CPUCreditBalance` est nulle. La valeur de `CPUSurplusCreditBalance` est remboursée progressivement par les crédits UC gagnés. Si le nombre de crédits excédentaires dépasse le nombre maximum de crédits que le nœud peut gagner en 24 heures, les crédits excédentaires dépensés au-dessus du maximum génèrent des frais supplémentaires. Unités `Credits (vCPU-minutes)` : Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrique	Description
`CPUSurplusCreditsCharged`	Nombre de crédits excédentaires dépensés qui ne sont pas remboursés progressivement par les crédits UC gagnés et qui génèrent donc des frais supplémentaires. Des crédits excédentaires sont facturés quand les crédits excédentaires dépensés dépassent le nombre maximum de crédits que le nœud peut gagner sur une période de 24 heures. Les crédits excédentaires dépensés au-delà de ce maximum sont facturés à la fin de l'heure ou quand le nœud est résilié. Unités `Credits (vCPU-minutes)` : Statistiques valides : • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> Les métriques `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance` et `CPUSurplusCreditsCharged` sont disponibles uniquement pour les nœuds T3.

Dimensions pour les métriques DAX

Les métriques pour DAX sont qualifiées par les valeurs pour le compte, l'ID de cluster, ou l'ID de cluster et l'ID de nœud. Vous pouvez utiliser la CloudWatch console pour récupérer des données DAX selon l'une des dimensions du tableau suivant.

Dimension	Description
`Account`	Fournit des statistiques agrégées sur tous les nœuds dans un compte.
`ClusterId`	Limite les données à un cluster.
`ClusterId, NodeId`	Limite les données à un nœud au sein d'un cluster.

Création d' CloudWatch alarmes pour surveiller le DAX

Vous pouvez créer une CloudWatch alarme Amazon qui envoie un message Amazon Simple Notification Service (Amazon SNS) lorsque l'état de l'alarme change. Une alarme surveille une seule métrique pendant la période que vous spécifiez. Elle réalise une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné sur un certain nombre de périodes. L'action est une notification envoyée à une rubrique Amazon SNS ou à une politique de scalabilité automatique. Les alarmes déclenchent des actions uniquement pour les changements d'état prolongés. CloudWatch les alarmes n'appellent pas d'actions simplement parce qu'elles sont dans un état particulier. L'état doit avoir changé et avoir été maintenu pendant un nombre de périodes spécifié.

Comment puis-je être informé des échecs de cache de requête ?

1. Créez une rubrique Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

   Pour plus d'informations, consultez Configurer Amazon Simple Notification Service dans le guide de CloudWatch l'utilisateur Amazon.

2. Créez l'alerte.

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. Testez l'alarme.

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> Vous pouvez augmenter ou diminuer le seuil de manière à répondre aux besoins de votre application. Vous pouvez également utiliser CloudWatch Metric Math pour définir une métrique du taux d'échec du cache et définir une alarme sur cette métrique.

Comment puis-je être informé si des demandes entraînent une erreur interne dans le cluster ?

1. Créez une rubrique Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

   Pour plus d'informations, consultez Configurer Amazon Simple Notification Service dans le guide de CloudWatch l'utilisateur Amazon.

2. Créez l'alerte.

```
aws cloudwatch put-metric-alarm \
```

```
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Testez l’alarme.

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

Surveillance en production

Vous avez tout intérêt à établir une base de référence en ce qui concerne les performances normales de DAX dans votre environnement, en mesurant les performances à divers moments et dans diverses conditions de charge. Lorsque vous surveillez DAX, songez à stocker l’historique des données de surveillance. Ces données stockées constituent une référence pour comparer avec des données de performances actuelles, identifier les modèles de performance normaux et les anomalies de performance, et concevoir des méthodes pour résoudre les problèmes.

Pour établir une base de référence, vous devez au minimum surveiller les éléments suivants pendant le test de charge et en production :

- Utilisation de l’UC et demandes de limitation, afin que vous puissiez déterminer si vous avez besoin d’utiliser un type de nœud plus important dans votre cluster. L'utilisation du processeur de votre cluster est disponible via la `CPUUtilization` CloudWatch métrique. Les statistiques moyennes de cette métrique fournissent une vue de l’utilisation moyenne du processeur sur tous les nœuds de votre cluster. Pour les décisions relatives à la mise à l’échelle du cluster, nous vous recommandons d’utiliser la statistique maximale, qui est l’utilisation maximale sur tous les nœuds.

> **Note**
>
> AWS a amélioré la granularité de la `CPUUtilization` métrique. Il se peut que vous observiez des modifications de la métrique entre le 17 mai 2024 et le 22 juin 2024.

- La latence d'opération (mesurée côté client) doit rester homogène dans les conditions de latence exigées par votre application.
- Les taux d'erreur devraient rester faibles, comme le montrent les `FailedRequestCount` CloudWatch indicateurs `ErrorRequestCount``FaultRequestCount`, et.
- Utilisation des octets réseau, afin que vous puissiez déterminer si vous devez utiliser un plus grand nombre de nœuds ou un type de nœud plus important dans votre cluster. Pour surveiller la consommation, vous pouvez définir des alertes `BaselineNetworkBytesInUtilization` et `BaselineNetworkBytesOutUtilization` des métriques disponibles dans CloudWatch, qui indiquent le pourcentage de consommation de bande passante réseau disponible pour votre type d'instance, pour le trafic entrant et sortant respectivement.
- L'utilisation de la mémoire cache et la quantité de données expulsées, afin de pouvoir déterminer si le type de nœud du cluster a suffisamment de mémoire pour contenir votre ensemble de travail. Si ce n'est pas le cas, basculez vers un type de nœud plus grand.

> **Note**
>
> Lorsque le nombre d'échecs de cache et d'écritures est important, l'utilisation de la mémoire cache peut augmenter jusqu'à 100 % et entraîner des interruptions de disponibilité.

- Les connexions client, de sorte que vous puissiez surveiller les pics inexpliqués dans les connexions au cluster.

Journalisation des opérations DAX avec AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) est intégréAWS CloudTrailà un service qui fournit un enregistrement des actions effectuées par un utilisateur, un rôle ou AWS un service dans DAX.

Pour en savoir plus sur DAX et CloudTrail, consultez la section Accélérateur DynamoDB (DAX) dans. Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail

Instances extensibles DAX T3/T2

DAX vous permet de choisir entre des instances à performances fixes (telles que R4, R5 et R7) et des instances à performances extensibles (telles que T2 et T3). Les instances à capacité extensible fournissent un niveau de base de performances d'UC, avec la possibilité de dépasser ce niveau en cas de besoin.

Les performances de base et la possibilité de les dépasser sont régies par des crédits UC. Les instances à capacité extensible accumulent des crédits UC en continu, à un rythme déterminé par la taille d'instance, lorsque la charge de travail est inférieure au seuil de base. Ces crédits peuvent ensuite être consommés lorsque la charge de travail augmente. Un crédit UC fournit les performances d'un cœur UC complet pendant une minute.

Bon nombre de charges de travail n'ont pas besoin de niveaux d'UC élevés en permanence, mais tirent considérablement parti de l'accès complet à des UC très rapides quand elles en ont besoin. Les instance à performances extensibles sont spécifiquement conçues pour ces cas d'utilisation. Si vous avez besoin en permanence de performances d'UC élevées pour votre base de données, nous vous recommandons d'utiliser des instances à capacité fixe.

Famille d'instances DAX T2

Les instances DAX T2 sont des instances aux performances polyvalentes qui fournissent un niveau de base de performances d'UC, avec la possibilité de dépasser ce niveau en cas de besoin. Les instances T2 constituent un bon choix pour les charges de travail de test et de développement qui nécessitent une prévisibilité des prix. Les instances DAX T2 sont configurées pour le mode standard. Cela signifie que, si l'instance vient à manquer de crédits accumulés, son utilisation d'UC diminue progressivement jusqu'à atteindre le niveau de base. Pour plus d'informations sur le mode standard, consultez Standard mode for burstable performance instances dans le Guide de l'utilisateur Amazon EC2.

Famille d'instances DAX T3

Les instances T3 sont le type d'instance à usage général extensible de nouvelle génération qui fournit un niveau de performances d'UC de base avec la possibilité d'étendre l'utilisation de l'UC à tout moment et aussi longtemps que nécessaire. Les instances T3 établissent un équilibre entre les ressources de calcul, de mémoire et de réseau, et sont idéales pour des charges de travail nécessitant une utilisation modérée de l'UC, qui sont sujettes à des pics d'utilisation temporaires. Les instances DAX T3 sont configurées pour le mode illimité. Cela signifie qu'elles peuvent

dépasser le niveau de base d'utilisation de l'UC sur une période de 24 heures moyennant des frais supplémentaires. Pour plus d'informations sur le mode illimité, consultez Unlimited mode for burstable performance instances dans le Guide de l'utilisateur Amazon EC2.

Les instances DAX T3 peuvent soutenir des performances UC élevées aussi longtemps qu'une charge de travail l'exige. Pour la majorité des charges de travail polyvalentes, les instances T3 fournissent d'excellentes performances sans frais supplémentaires. Le prix horaire d'une instance T3 couvre automatiquement tous les pics d'utilisation temporaires quand l'utilisation moyenne de l'UC est égale ou inférieure au niveau de base sur une période de 24 heures.

Par exemple, une instance `dax.t3.small` reçoit des crédits en continu au rythme de 24 crédits UC par heure. Cette capacité fournit des performances de base équivalant à 20 % d'un cœur de processeur (20 % × 60 minutes = 12 minutes). Si l'instance n'utilise pas les crédits qu'elle reçoit, ceux-ci sont stockés dans son solde de crédits UC jusqu'à un maximum de 576 crédits UC. Quand l'instance `t3.small` doit dépasser le niveau de 20 % d'utilisation de la capacité d'un cœur, elle puise dans son solde de crédits UC pour gérer automatiquement cette surtension.

Si les instances DAX T2 sont limitées aux performances de base une fois le solde de crédits UC réduit à zéro, les instances DAX T3 peuvent dépasser le niveau d'utilisation de base même lorsque leur solde de crédits UC est nul. Pour la grande majorité des charges de travail où l'utilisation moyenne de l'UC est égale ou inférieure au niveau des performances de base, le prix horaire de base pour `t3.small` couvre tous les dépassements du niveau d'utilisation de base de l'UC. Si l'instance opère en moyenne à 25 % d'utilisation de l'UC (5 % au-dessus du niveau de base) sur une période de 24 heures après que son solde de crédits UC est réduit à zéro, elle est facturée 11,52 cents supplémentaires (9,6 cents/vCPU-heure × 1 vCPU × 5 % × 24 heures). Pour plus de détails sur la tarification, consultez Tarification Amazon DynamoDB.

Contrôle d'accès à DAX

DynamoDB Accelerator (DAX) est conçu pour fonctionner avec DynamoDB, afin d'ajouter en toute fluidité une couche de mise en cache à vos applications. Toutefois, DAX et DynamoDB ont des mécanismes de contrôle d'accès distincts. Les deux services utilisent Gestion des identités et des accès AWS(IAM) pour implémenter leurs politiques de sécurité respectives, mais les modèles de sécurité pour DAX et DynamoDB sont différents.

Nous vous recommandons vivement de vous familiariser avec les deux modèles de sécurité. Vous pourrez ainsi implémenter des mesures de sécurité appropriées pour vos applications qui utilisent DAX.

Cette section décrit les mécanismes de contrôle d'accès fournis par DAX et propose des exemples de politiques IAM que vous pouvez personnaliser en fonction de vos besoins.

Avec DynamoDB, vous pouvez créer des politiques IAM qui limitent les actions qu'un utilisateur peut effectuer sur des ressources DynamoDB individuelles. Par exemple, vous pouvez créer un rôle d'utilisateur qui ne permet à l'utilisateur d'effectuer des actions en lecture seule que sur une table DynamoDB déterminée. (Pour plus d'informations, consultez Gestion des identités et des accès pour Amazon DynamoDB.) En comparaison, le modèle de sécurité DAX est axé sur la sécurité du cluster et sur la capacité de celui-ci à effectuer pour vous des actions d'API DynamoDB.

> **Warning**
>
> Si vous utilisez actuellement des rôles et politiques IAM pour limiter l'accès aux données des tables DynamoDB, l'utilisation de DAX peut contrecarrer ces politiques. Par exemple, un utilisateur peut avoir accès à une table DynamoDB via DAX mais ne pas bénéficier d'un accès explicite à cette même table en accédant directement à DynamoDB. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour Amazon DynamoDB.
>
> DAX n'applique pas de séparation au niveau de l'utilisateur sur les données dans DynamoDB. Au lieu de cela, les utilisateurs héritent des autorisations de la politique IAM du cluster DAX quand ils accèdent à ce dernier. Autrement dit, au moment d'accéder à des tables DynamoDB via DAX, les seuls contrôles d'accès en vigueur sont les autorisations dans la politique IAM du cluster DAX. Aucune autre autorisation n'est reconnue.
>
> Si une isolation est requise, nous vous recommandons de créer des clusters DAX supplémentaires et de définir la portée de la politique IAM de chaque cluster en conséquence. Par exemple, vous pouvez créer plusieurs clusters DAX et autoriser chacun d'eux à accéder à une seule table.

Fonction du service IAM pour DAX

Lorsque vous créez un cluster DAX, vous devez l'associer à un rôle IAM. C'est ce qui s'appelle le rôle de service du cluster.

Supposons que vous souhaitiez créer un nouveau cluster DAX nommé DAXCluster01. Vous pouvez créer un rôle de service nommé DAXService Role et l'associer à DAXCluster01. La politique relative à DAXServiceRole définirait les actions DynamoDB DAXClusterque 01 pourrait effectuer pour le compte des utilisateurs qui interagissent avec 01. DAXCluster

Lorsque vous créez un rôle de service, vous devez spécifier une relation de confiance entre le DAXServicerôle et le service DAX lui-même. Une relation d'approbation détermine les entités qui peuvent endosser un rôle et utiliser ses autorisations. Voici un exemple de document de relation de confiance pour DAXServiceRole :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Cette relation de confiance permet à un cluster DAX d'assumer un DAXServicerôle et d'effectuer des appels d'API DynamoDB en votre nom.

Les actions d'API DynamoDB autorisées sont décrites dans un document de politique IAM, que vous joignez à Role. DAXService Voici un exemple de document politique :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

Cette politique permet à DAX d'exécuter toutes les actions d'API DynamoDB sur une table DynamoDB. L'action `dynamodb:DescribeTable` est requise pour que DAX puisse gérer les métadonnées sur la table, tandis que les autres actions sont celles de lecture et d'écriture effectuées sur des éléments de la table. La table, nommée `Books`, se trouve dans la région us-west-2 et est détenue par l'ID de compte AWS `123456789012`.

> **Note**
>
> DAX prend en charge des mécanismes pour prévenir le problème d'adjoint confus lors de l'accès interservice. Pour de plus amples informations, veuillez consulter Le problème du député confus dans le Guide de l'utilisateur IAM.

Politique IAM pour autoriser l'accès au cluster DAX

Après avoir créé un cluster DAX, vous devez accorder des autorisations à un utilisateur pour lui permettre d'accéder au cluster DAX.

Supposons, par exemple, que vous souhaitiez accorder l'accès à DAXCluster01 à une utilisatrice nommée Alice. Vous devez d'abord créer une politique IAM (AliceAccessPolicy) qui définit les clusters DAX et les actions d'API DAX auxquels le destinataire peut accéder. Vous devez ensuite octroyer l'accès en attachant cette stratégie à l'utilisatrice Alice.

Le document de politique suivant donne au destinataire un accès complet au DAXCluster01.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Si le document de politique autorise l'accès au cluster DAX, il n'accorde aucune autorisation DynamoDB. (Les autorisations DynamoDB sont conférées par le rôle de service DAX)

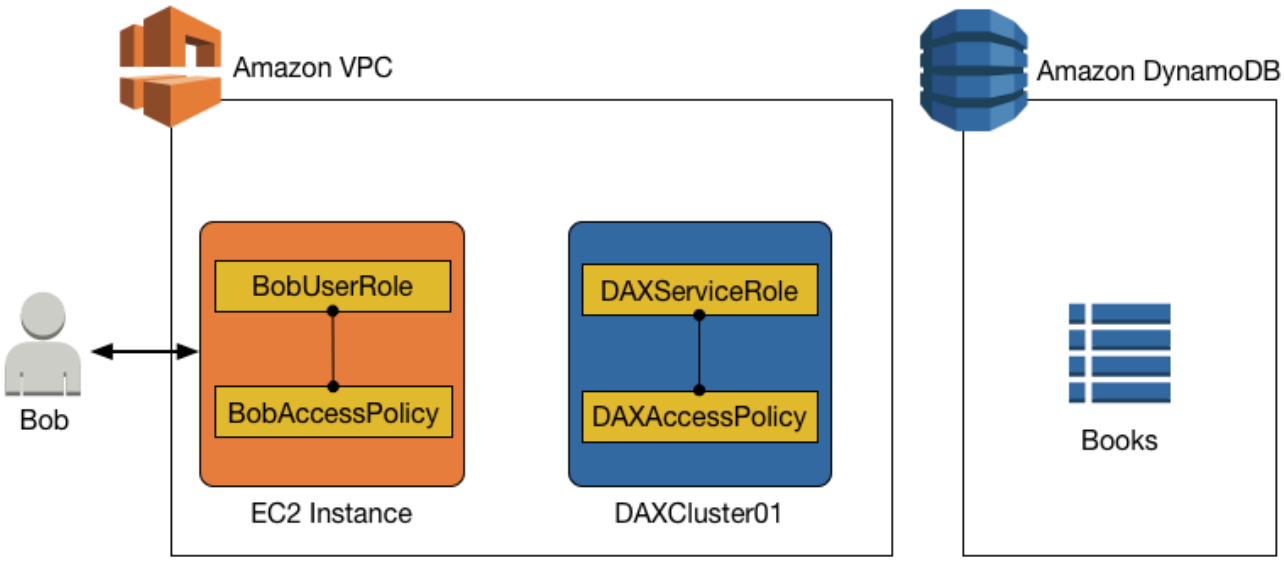
Pour l'utilisatrice Alice, vous devez d'abord créer `AliceAccessPolicy` avec le document de stratégie présenté ci-dessus. Vous devez ensuite attacher la stratégie à Alice.

> **Note**
>
> Au lieu d'attacher la politique à un utilisateur, vous pouvez l'attacher à un rôle IAM. De cette manière, tous les utilisateurs qui endossent ce rôle bénéficient des autorisations que vous avez définies dans la stratégie.

La politique utilisateur, en combinaison avec le rôle de service DAX, détermine les ressources DynamoDB et les actions d'API auxquelles la destinataire peut accéder via DAX.

Étude de cas : accès à DynamoDB et à DAX

Le scénario suivant permet de mieux comprendre les politiques IAM à utiliser avec DAX. (Des allusions seront faites à ce scénario tout au long de cette section.) Le schéma suivant offre un aperçu général du scénario.



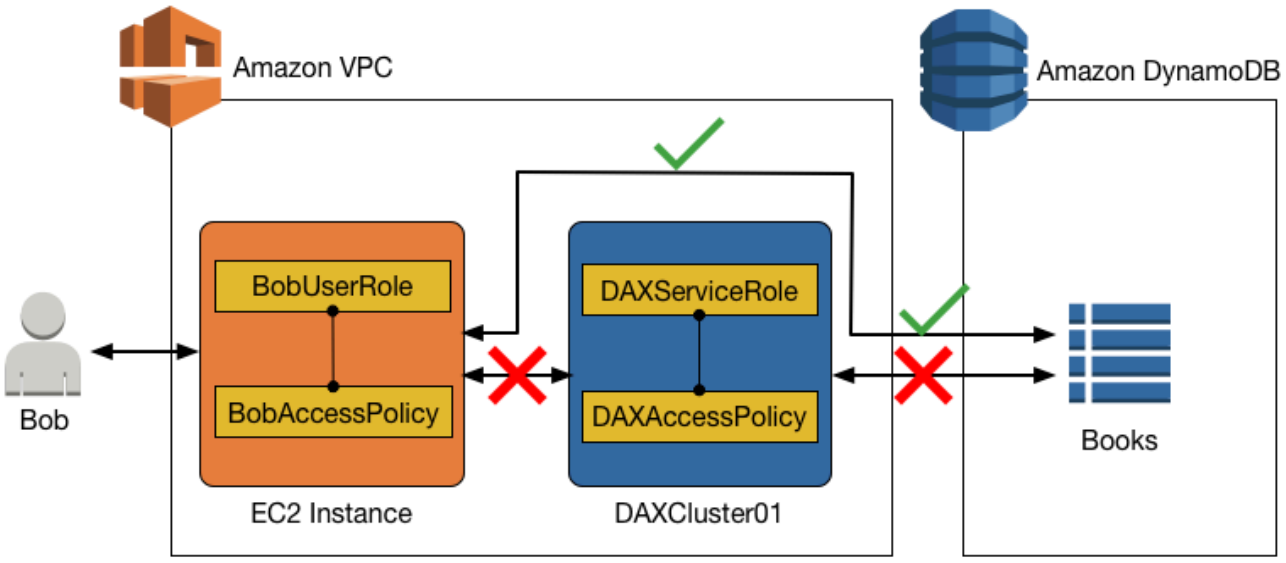
Ce scénario réunit les entités suivantes :

- Un utilisateur (Bob).
- Un rôle IAM (`BobUserRole`). Bob endosse ce rôle au moment de l'exécution.
- Une politique IAM (`BobAccessPolicy`). Cette politique est attachée à `BobUserRole`. `BobAccessPolicy` définit les ressources DynamoDB et DAX auxquelles `BobUserRole` est autorisé à accéder.
- Un cluster DAX (`DAXCluster01`).
- Un rôle de service IAM (`DAXServiceRole`). Ce rôle permet à `DAXCluster01` d'accéder à DynamoDB.
- Une politique IAM (`DAXAccessPolicy`). Cette politique est jointe à`DAXServiceRole`. `DAXAccessPolicy`définit le APIs DynamoDB et les ressources `DAXCluster01` auxquelles l'accès est autorisé.
- Une table DynamoDB (`Books`).

La combinaison des déclarations de stratégie dans `BobAccessPolicy` et `DAXAccessPolicy` détermine ce que Bob peut faire avec la table `Books`. Par exemple, Bob peut être autorisé à accéder à `Books` directement (via le point de terminaison DynamoDB), indirectement (via le cluster DAX) ou

les deux à la fois. Bob peut aussi être autorisé à lire les données de `Books`, à écrire des données dans `Books` ou les deux à la fois.

Accès à DynamoDB, mais pas d'accès avec DAX



Il est possible d'autoriser un accès direct à une table DynamoDB tout en empêchant tout accès indirect à l'aide d'un cluster DAX. Pour un accès direct à DynamoDB, les autorisations pour le rôle `BobUserRole` sont déterminées par la politique `BobAccessPolicy` (attachée au rôle).

Accès en lecture seule à DynamoDB (uniquement)

Bob peut accéder à DynamoDB avec le rôle `BobUserRole`. La politique IAM associée à ce rôle (`BobAccessPolicy`) détermine les tables DynamoDB auxquelles il est possible d'accéder et `BobUserRole` APIs celles qu'elles peuvent invoquer. `BobUserRole`

Penchons-nous sur le document de stratégie suivant pour `BobAccessPolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Lorsque ce document est attaché à `BobAccessPolicy`, il autorise `BobUserRole` à accéder au point de terminaison DynamoDB et à effectuer des opérations en lecture seule sur la table `Books`.

DAX n'apparaissant pas dans cette politique, l'accès via DAX est rejeté.

Accès en lecture-écriture à DynamoDB (uniquement)

Si read/write l'accès à DynamoDB est `BobUserRole` requis, la politique suivante fonctionnera.

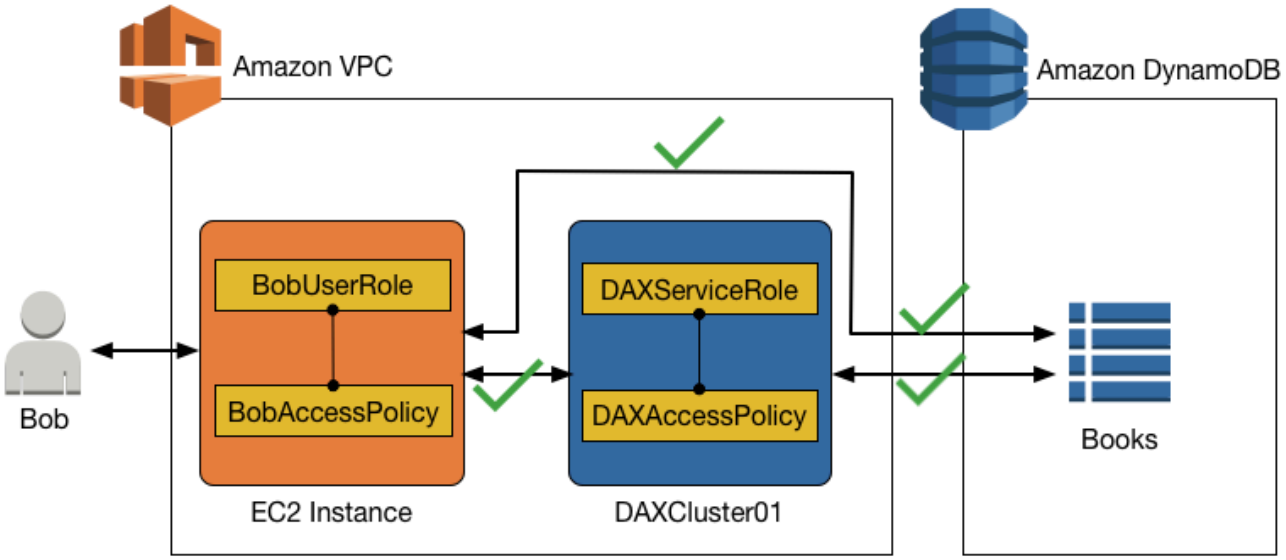
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

A nouveau, DAX n'apparaissant pas dans cette politique, l'accès via DAX est rejeté.

Accès à DynamoDB et à DAX



Pour autoriser l'accès à un cluster DAX, vous devez inclure des actions spécifiques de DAX dans une politique IAM.

Les actions spécifiques de DAX correspondent à leurs homologues de même nom dans l'API DynamoDB :

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`

- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

Il en va de même pour la clé de condition `dax:EnclosingOperation`.

Accès en lecture seule à DynamoDB et à DAX

Supposons que Bob a besoin d'un accès en lecture seule à la table `Books`, à partir de DynamoDB et de DAX. La stratégie suivante (attachée à `BobUserRole`) confèrerait cet accès.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

La politique comporte une instruction pour l'accès DAX (`DAXAccessStmt`) et une autre pour Dynamo DBaccess (`DynamoDBAccessStmt`). Ces déclarations permettraient à Bob d'envoyer des demandes `GetItem`, `BatchGetItem`, `Query`et `Scan` à `DAXCluster01`.

Toutefois, le rôle de service pour `DAXCluster01` nécessiterait également un accès en lecture seule à la table `Books` dans DynamoDB. La politique IAM suivante, attachée à `DAXServiceRole`, répondrait à cette exigence.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Accès en lecture-écriture à DynamoDB et en lecture seule avec DAX

Pour un rôle d'utilisateur donné, vous pouvez donner read/write accès à une table DynamoDB, tout en autorisant l'accès en lecture seule via DAX.

Pour Bob, la politique IAM pour `BobUserRole` doit autoriser les actions de lecture et d'écriture de DynamoDB sur la table `Books`, tout en prenant également en charge les actions en lecture seule via `DAXCluster01`.

L'exemple de document de stratégie suivant confèrerait cet accès à `BobUserRole`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Par ailleurs, `DAXServiceRole` aurait besoin d'une politique IAM permettant à `DAXCluster01` d'effectuer des actions en lecture seule sur la table `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Read/write access to DynamoDB and read/writeaccès au DAX

Supposons maintenant que Bob ait besoin read/write d'accéder à la `Books` table, directement depuis DynamoDB ou indirectement depuis. `DAXCluster01` La stratégie suivante (attachée à `BobAccessPolicy`, confèrerait cet accès.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
```

```
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

En outre, `DAXServiceRole` cela nécessiterait une politique IAM permettant d'`DAXCluster01`effectuer des read/write actions sur la `Books` table.

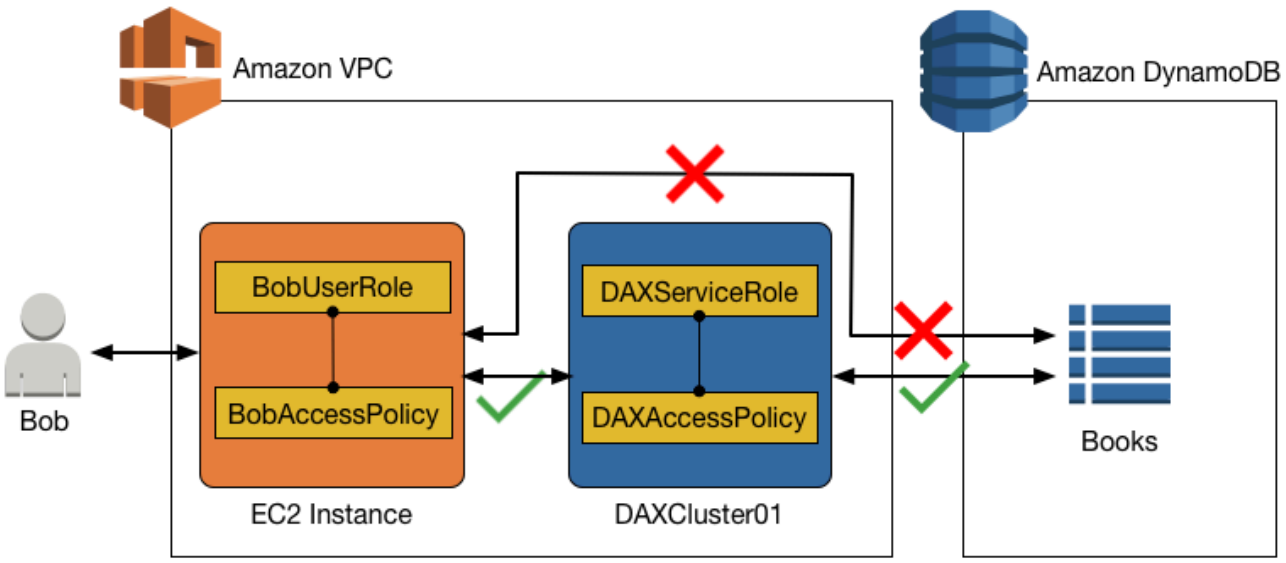
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Accès à DynamoDB via DAX, mais aucun accès direct à DynamoDB

Dans ce scénario, Bob peut accéder à la table `Books` via DAX, mais il n'a pas d'accès direct à la table `Books` dans DynamoDB. Par conséquent, lorsque Bob a accès à DAX, il a également accès à une table DynamoDB à laquelle il n'aurait autrement pas accès. Lorsque vous configurez une politique IAM pour le rôle de service DAX, n'oubliez pas qu'un utilisateur ayant accès au cluster DAX en vertu de la politique d'accès utilisateur a également accès aux tables spécifiées dans celle-ci. Dans ce cas, `BobAccessPolicy` bénéficie d'un accès aux tables spécifiées dans `DAXAccessPolicy`.



Si vous utilisez actuellement des rôles et des politiques IAM pour limiter l'accès aux tables et aux données DynamoDB, l'utilisation de DAX peut contrecarrer ces politiques. Dans la politique suivante, Bob a accès à une table DynamoDB via DAX, mais il ne bénéficie pas d'un accès direct explicite à cette même table dans DynamoDB.

Le document de stratégie suivant (`BobAccessPolicy`), attaché à `BobUserRole`, confèrerait cet accès.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

Cette politique d'accès n'inclut aucune autorisation d'accès direct à DynamoDB.

Avec `BobAccessPolicy`, la politique `DAXAccessPolicy` suivante accorde à `BobUserRole` l'accès à la table DynamoDB `Books`, même si `BobUserRole` ne peut pas accéder directement à la table `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Comme le montre cet exemple, lorsque vous configurez le contrôle d'accès pour la politique d'accès utilisateur et la politique d'accès au cluster DAX, vous devez bien comprendre l' end-to-endaccès afin de garantir le respect du principe du moindre privilège. Assurez-vous également que le fait d’accorder à un utilisateur l’accès à un cluster DAX ne va pas à l’encontre de politiques de contrôle d’accès définies antérieurement.

Chiffrement au repos DAX

Le chiffrement Amazon DynamoDB Accelerator (DAX) au repos fournit une couche supplémentaire de protection des données en contribuant à sécuriser vos données contre tout accès non autorisé au stockage sous-jacent. Les politiques organisationnelles et les réglementations sectorielles ou gouvernementales, ainsi que les exigences de conformité, peuvent nécessiter l’utilisation du chiffrement au repos pour protéger vos données. Vous pouvez utiliser le chiffrement pour renforcer la sécurité des données de vos applications déployées dans le cloud.

Avec le chiffrement au repos, les données conservées par DAX sur disque sont chiffrées en utilisant les algorithmes Advanced Encryption Standard 256 bits, également appelé chiffrement AES-256. DAX écrit les données sur disque dans le cadre de la propagation des modifications du nœud principal aux nœuds de réplica en lecture.

Le chiffrement DAX au repos est automatiquement intégré avec AWS Key Management Service (AWS KMS) pour gérer la clé de service par défaut unique utilisée pour chiffrer vos clusters. S'il n'existe pas de clé de service par défaut lorsque vous créez votre cluster DAX chiffré, AWS KMS crée automatiquement une clé gérée par AWS pour vous. Cette clé sera utilisée avec les clusters chiffrés qui seront dans le futur. AWS KMS combine du matériel et des logiciels sécurisés et hautement disponibles pour fournir un système de gestion de clés à l'échelle du cloud.

Une fois vos données chiffrées, DAX gère leur déchiffrement de façon transparente avec un impact minimal sur les performances. Vous n'avez pas besoin de modifier vos applications pour utiliser le chiffrement.

> **Note**
>
> DAX n'appelle pas AWS KMS pour chaque opération DAX unique. DAX n'utilise la clé qu'au lancement du cluster. Même si l'accès est révoqué, DAX peut toujours accéder aux données jusqu'à ce que le cluster soit arrêté. Les clés AWS KMS spécifiées par le client ne sont pas prises en charge.

Le chiffrement DAX au repos est disponible pour les types de nœuds de cluster suivants :

Famille	Type de nœud
Mémoire optimisée (R4, R5 et R7)	dax.r4.large
	dax.r4.xlarge
	dax.r4.2xlarge
	dax.r4.4xlarge
	dax.r4.8xlarge
	dax.r4.16xlarge

Famille	Type de nœud
	dax.r5.large dax.r5.xlarge dax.r5.2xlarge dax.r5.4xlarge dax.r5.8xlarge dax.r5.12xlarge dax.r5.16xlarge dax.r5.24xlarge dax.r7i.large dax.r7i.xlarge dax.r7i.2xlarge dax.r7i.4xlarge dax.r7i.8xlarge dax.r7i.12xlarge dax.r7i.16xlarge dax.r7i.24xlarge
Usage général (T2)	dax.t2.small dax.t2.medium
Usage général (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> Le chiffrement DAX au repos n'est pas pris en charge pour les types de nœuds `dax.r3.*`.

Le chiffrement au repos ne peut pas être activé ni désactivé après la création d'un cluster. Vous devez recréer le cluster pour activer le chiffrement au repos s'il n'a pas été activé à la création.

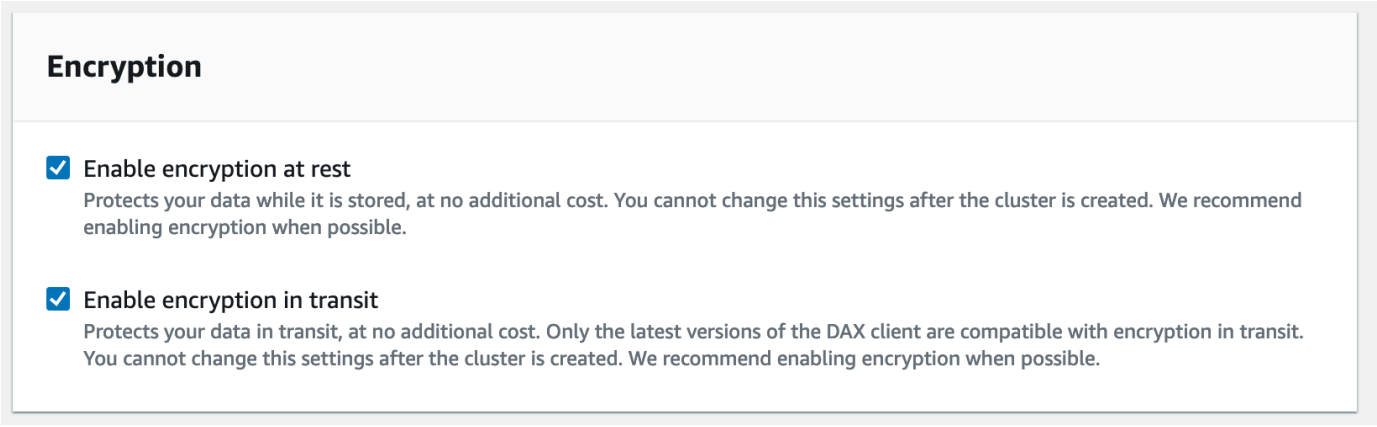
Le chiffrement DAX au repos est offert sans coût supplémentaire (des coûts d'utilisation de clé de chiffrement AWS KMS s'appliquent). Pour plus d'informations sur la tarification, consultez Tarification Amazon DynamoDB.

Activation du chiffrement au repos à l'aide de la AWS Management Console

Suivez ces étapes pour activer le chiffrement DAX au repos sur une table à l'aide de la console.

Pour activer le chiffrement DAX au repos

1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, sous DAX, choisissez Clusters.
3. Choisissez Créer un cluster.
4. Pour le Cluster name (Nom du cluster), saisissez un nom court pour votre cluster. Choisissez le type de nœud pour tous les nœuds du cluster et utilisez **3** nœuds pour la taille du cluster.
5. Dans Chiffrement, assurez-vous que Activer le chiffrement est sélectionné.



6. Après avoir choisi le rôle IAM, le groupe de sous-réseaux, les groupes de sécurité et les paramètres du cluster, choisissez Lancer le cluster.

Pour vérifier que le cluster est chiffré, consultez les détails du cluster sous le volet Clusters. Le chiffrement doit être paramétré sur ENABLED (ACTIVÉ).

Chiffrement DAX en transit

Amazon DynamoDB Accelerator (DAX) prend en charge le chiffrement de données en transit entre votre application et votre cluster DAX, ce qui vous permet d'utiliser DAX dans des applications imposant des exigences de chiffrement strictes.

Que vous choisissiez ou non le chiffrement en transit, le trafic entre votre application et votre cluster DAX reste dans votre VPC Amazon. Ce trafic est acheminé vers des interfaces réseau Elastic à l'aide des adresses IP privées dans votre VPC, qui sont attachées aux nœuds de votre cluster. Avec votre VPC comme délimitation d'approbation, vous disposez d'un contrôle conséquent sur la sécurité de vos données grâce à l'utilisation d'outils standard tels que les groupes de sécurité, la segmentation de sous-réseau à l'aide d'ACL réseau et le suivi de flux VPC. Le chiffrement en transit de DAX s'ajoute à ce niveau de confidentialité de base, garantissant que toutes les demandes et réponses entre l'application et le cluster sont chiffrées par TLS (Transport Level Security), et que les connexions au cluster peuvent être authentifiées par vérification d'un certificat x509 de cluster. Les données que DAX écrit sur disque peuvent également être chiffrées si vous optez pour le chiffrement au repos lors de la création de votre cluster DAX.

L'utilisation du chiffrement en transit avec DAX est facile. Sélectionnez simplement cette option lors de la création d'un cluster, et utilisez une version récente de l'un des clients DAX dans votre application. Les clusters qui utilisent le chiffrement en transit ne prenant pas en charge le trafic non chiffré, il est impossible de mal configurer votre application et de contourner le chiffrement. Le client DAX utilise le certificat x509 du cluster pour authentifier l'identité du cluster lors de l'établissement de connexions, en s'assurant que vos demandes DAX arrivent à destination. Toutes les méthodes de création de clusters DAX prennent en charge le chiffrement en transit : l'AWS Management Console, l'AWS CLI, tous les kits SDK et CloudFormation.

Le chiffrement en transit ne peut pas être activé sur un cluster DAX existant. Pour utiliser le chiffrement en transit dans une application DAX existante, créez un cluster avec le chiffrement en transit activé, réacheminez le trafic de votre application vers celui-ci, puis supprimez l'ancien cluster.

Utilisation des rôles IAM liés à un service pour DAX

Amazon DynamoDB Accelerator (DAX) utilise des rôles liés à un service Gestion des identités et des accès AWS (IAM). Un rôle lié à un service est un type unique de rôle IAM lié directement à DAX.

Les rôles liés à un service sont prédéfinis par DAX et incluent toutes les autorisations que le service requiert pour appeler d'autres services AWS en votre nom.

Un rôle lié à un service simplifie la configuration de DAX, car vous n'avez pas besoin d'ajouter manuellement les autorisations requises. DAX définit les autorisations de ses rôles liés à un service. Sauf spécification contraire, seul DAX peut endosser ses rôles. Les autorisations définies comprennent la politique d'approbation et la politique d'autorisations. Cette politique d'autorisations ne peut pas être attachée à une autre entité IAM.

Vous pouvez supprimer les rôles uniquement après la suppression préalable de leurs ressources connexes. Vos ressources DAX sont ainsi protégées, car vous ne pouvez pas supprimer par inadvertance l'autorisation d'accéder aux ressources.

Pour obtenir des informations sur d'autres services qui prennent en charge les rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM dans le Guide de l'utilisateur IAM. Recherchez les services qui comportent un Yes (Oui) dans la colonne Service-linked roles (Rôles liés à un service). Choisissez un lien Yes (Oui) pour consulter la documentation du rôle lié à ce service.

Rubriques

- Autorisations des rôles liés à un service pour DAX
- Création d'un rôle lié à un service pour DAX
- Modification d'un rôle lié à un service pour DAX
- Suppression d'un rôle lié à un service pour DAX

Autorisations des rôles liés à un service pour DAX

DAX utilise le rôle lié à un service nommé `AWSServiceRoleForDAX`. Ce rôle permet à DAX d'appeler des services au nom de votre cluster DAX.

> **Important**
>
> Le rôle lié à un service `AWSServiceRoleForDAX` facilite la configuration et la maintenance d'un cluster DAX. Cependant, vous devez continuer d'accorder à chaque cluster l'accès à DynamoDB avant de pouvoir l'utiliser. Pour de plus amples informations, consultez Contrôle d'accès à DAX.

Le rôle lié à un service `AWSServiceRoleForDAX` approuve les services suivants pour endosser le rôle :

- `dax.amazonaws.com`

La politique d'autorisations de rôle permet à DAX d'exécuter les actions suivantes sur les ressources spécifiées :

- Actions sur `ec2` :
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`
  - `RevokeSecurityGroupIngress`

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Pour plus d'informations, consultez Service-Linked Role Permissions (autorisations du rôle lié à un service) dans le Guide de l'utilisateur IAM.

Pour permettre à une entité IAM de créer des rôles liés à un service AWSServiceRoleForDAX

Ajoutez la déclaration de stratégie suivante aux autorisations de cette entité IAM.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
```

```
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Création d'un rôle lié à un service pour DAX

Vous n'avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous créez un cluster, DAX crée automatiquement le rôle lié à un service pour vous.

> **Important**
>
> Si vous utilisiez le service DAX avant le 28 février 2018, quand il a commencé à prendre en charge les rôles liés à un service, DAX a créé le rôle `AWSServiceRoleForDAX` dans votre compte. Pour plus d'informations, consultez Un nouveau rôle est apparu dans mon compte AWS dans le Guide de l'utilisateur IAM.

Si vous supprimez ce rôle lié à un service et que vous devez ensuite le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous créez une instance ou un cluster, DAX crée de nouveau le rôle lié à un service pour vous.

Modification d'un rôle lié à un service pour DAX

DAX ne vous permet pas de modifier le rôle lié à un service `AWSServiceRoleForDAX`. Une fois que vous avez créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence à ce rôle. Néanmoins, vous pouvez modifier la description du rôle à l'aide d'IAM. Pour plus d'informations, consultez Modification d'un rôle lié à un service dans le Guide de l'utilisateur IAM.

Suppression d'un rôle lié à un service pour DAX

Si vous n'avez plus besoin d'utiliser une fonctionnalité ou un service qui nécessite un rôle lié à un service, nous vous recommandons de supprimer ce rôle. De cette façon, vous n'avez aucune entité inutilisée qui n'est pas surveillée ou gérée activement. Cependant, vous devez supprimer tous vos clusters DAX avant de pouvoir supprimer le rôle lié à un service.

Nettoyage d'un rôle lié à un service

Avant de pouvoir utiliser IAM pour supprimer un rôle lié à un service, vous devez d'abord vérifier qu'aucune session n'est active pour le rôle et supprimer toutes les ressources utilisées par le rôle.

Pour vérifier si une session est active pour le rôle lié à un service dans la console IAM

1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation de la console IAM, sélectionnez Roles (Rôles). Ensuite, sélectionnez le nom (pas la case à cocher) du rôle AWSServiceRoleForDAX.
3. Sur la page Récapitulatif du rôle sélectionné, choisissez l'onglet Access Advisor.
4. Dans l'onglet Access Advisor, consultez l'activité récente pour le rôle lié à un service.

> **Note**
>
> Si vous ignorez si DAX utilise le rôle `AWSServiceRoleForDAX`, vous pouvez essayer de le supprimer. Si le service utilise le rôle, la suppression échoue et vous pouvez afficher les régions dans lesquelles le rôle est utilisé. Si le rôle est utilisé, vous devez supprimer vos clusters DAX avant de pouvoir le supprimer le rôle. Vous ne pouvez pas révoquer la session d'un rôle lié à un service.

Si vous souhaitez supprimer le rôle `AWSServiceRoleForDAX`, vous devez au préalable supprimer tous vos clusters DAX.

Suppression de tous vos clusters DAX

Utilisez l'une de ces procédures pour supprimer chacun de vos clusters DAX.

Pour supprimer un cluster DAX (console)

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le panneau de navigation, sous DAX, choisissez Clusters.
3. Choisissez Actions, puis Supprimer.
4. Dans la boîte de dialogue de confirmation Supprimer un cluster, choisissez Supprimer.

Pour supprimer un cluster DAX (AWS CLI)

Consultez delete-cluster dans la Référence des commandes de l'AWS CLI.

Pour supprimer un cluster DAX (API)

Consultez DeleteCluster dans la Référence d'API Amazon DynamoDB.

Suppression du rôle lié à un service

Pour supprimer manuellement le rôle lié à un service à l’aide d’IAM

Utilisez la console IAM, l'interface de ligne de commande IAM ou l'API IAM pour supprimer le rôle lié à un service `AWSServiceRoleForDAX`. Pour plus d’informations, veuillez consulter Suppression d’un rôle lié à un service dans le Guide de l’utilisateur IAM.

Accès au DAX sur plusieurs comptes AWS

Imaginez que vous disposez d'un cluster DynamoDB Accelerator (DAX) exécuté sur AWS un compte (compte A) et que le cluster DAX doit être accessible depuis une instance Amazon Elastic Compute Cloud (EC2Amazon) d'un AWS autre compte (compte B). Dans ce didacticiel, vous lancez une EC2 instance dans le compte B avec un rôle IAM depuis le compte B. Vous utilisez ensuite les informations d'identification de sécurité temporaires de l' EC2 instance pour assumer un rôle IAM depuis le compte A. Enfin, vous utilisez les informations de sécurité temporaires du rôle IAM dans le compte A pour passer des appels d'application via une connexion d'appairage Amazon VPC au cluster DAX du compte A. Pour effectuer ces tâches, vous aurez besoin d'un accès administratif sur les deux comptes.AWS

> **Important**
>
> Il n’est pas possible de faire accéder un cluster DAX à une table DynamoDB d’un autre compte.

Rubriques

- Configurer IAM
- Configuration d’un VPC
- Modifier le client DAX pour autoriser l’accès intercompte

Configurer IAM

1. Créez un fichier texte nommé `AssumeDaxRoleTrust.json` avec le contenu suivant, qui permet EC2 à Amazon de travailler en votre nom.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Dans le compte B, créez un rôle qu'Amazon EC2 pourra utiliser lors du lancement d'instances.

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. Créez un fichier texte nommé `AssumeDaxRolePolicy.json` avec le contenu suivant, qui permet au code exécuté sur l' EC2 instance du compte B d'assumer un rôle IAM dans le compte A. Remplacez-le *`accountA`* par l'identifiant réel du compte A.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
        }
    ]
}
```

4. Ajoutez cette stratégie au rôle que vous venez de créer.

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. Créez un profil d'instance pour autoriser les instances à utiliser le rôle.

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. Associez le rôle au profil d'instance.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. Créez un fichier texte nommé `DaxCrossAccountRoleTrust.json` avec le contenu suivant, ce qui permet au compte B d'assumer un rôle du compte A. Remplacez *`accountB`* par l'identifiant réel du compte B.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. Dans le compte A, créez le rôle que le compte B peut assumer.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Créez un fichier texte nommé `DaxCrossAccountPolicy.json` permettant l'accès au cluster DAX. *`dax-cluster-arn`*Remplacez-le par le nom de ressource Amazon (ARN) correct de votre cluster DAX.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. Dans le compte A, ajoutez la stratégie au rôle.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Configuration d'un VPC

1. Recherchez le groupe de sous-réseaux du cluster DAX du compte A. *`cluster-name`*Remplacez-le par le nom du cluster DAX auquel le compte B doit accéder.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. À l'aide de cela*subnet-group*, trouvez le VPC du cluster.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. À l'aide de cela*vpc-id*, trouvez le CIDR du VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. À partir du compte B, créez un VPC à l'aide d'un CIDR sans chevauchement différent de celui trouvé à l'étape précédente. Ensuite, créez au moins un sous-réseau. Vous pouvez utiliser l'assistant de création de VPC dans le AWS Management Console ou le. AWS CLI
5. À partir du compte B, demandez une connexion d'appairage au VPC du compte A, comme décrit dans Création et acceptation d'une connexion d'appairage de VPC. À partir du compte A, acceptez la connexion.
6. Dans le compte B, recherchez la table de routage du nouveau VPC. Remplacez *vpc-id* par l'ID du VPC que vous avez créé dans le compte B.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Ajoutez une route pour envoyer le trafic destiné au CIDR du compte A à la connexion d'appairage du VPC. N'oubliez pas de *user input placeholder* remplacer chacune par les valeurs correctes pour vos comptes.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. À partir du compte A, recherchez la table de routage du cluster DAX à l'aide de celle *vpc-id* que vous avez trouvée précédemment.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. À partir du compte A, ajoutez une route pour envoyer le trafic destiné au CIDR du compte B à la connexion d'appairage du VPC. Remplacez chacune *user input placeholder* par les valeurs correctes pour vos comptes.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. À partir du compte B, lancez une EC2 instance dans le VPC que vous avez créé précédemment. Attribuez-lui `AssumeDaxInstanceProfile`. Vous pouvez utiliser l'assistant de lancement dans le AWS Management Console ou le AWS CLI. Notez le groupe de sécurité de l'instance.

11. Dans le compte A, recherchez le groupe de sécurité qu'utilise le cluster DAX. N'oubliez pas de *cluster-name* remplacer par le nom de votre cluster DAX.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Mettez à jour le groupe de sécurité du cluster DAX pour autoriser le trafic entrant en provenance du groupe de sécurité de l' EC2 instance que vous avez créée dans le compte B. N'oubliez pas de *user input placeholders* remplacer le par les valeurs correctes pour vos comptes.

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

À ce stade, une application sur l' EC2 instance du compte B peut utiliser le profil d'instance pour assumer le `arn:aws:iam::`*`accountA-id`*`:role/DaxCrossAccountRole` rôle et utiliser le cluster DAX.

Modifier le client DAX pour autoriser l’accès intercompte

> **Note**
>
> AWS Security Token Service(AWS STS) les informations d'identification sont des informations d'identification temporaires. Certains clients gèrent l’actualisation automatiquement, tandis que d’autres nécessitent une logique supplémentaire pour actualiser les informations d’identification. Nous vous recommandons de suivre les instructions de la documentation appropriée.

Java

Cette section vous aide à modifier votre code client DAX existant pour autoriser l’accès DAX entre comptes. Si vous n’avez pas encore de code client DAX, vous pouvez trouver des exemples de codes fonctionnels dans le didacticiel Java et DAX.

1. Ajoutez les importations suivantes.

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Obtenez un fournisseur d'informations d'identification AWS STS et créez-en un objet client DAX. N'oubliez pas de *`user input placeholder`* remplacer chacune par les valeurs correctes pour vos comptes.

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();
```

```
STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
    .build();
```

.NET

Cette section vous aide à modifier votre code client DAX existant pour autoriser l'accès DAX entre comptes. Si vous n'avez pas encore de code client DAX, vous pouvez trouver des exemples de codes fonctionnels dans le didacticiel .NET et DAX.

1. Ajoutez le AWSSDK. SecurityToken NuGet empaqueter vers la solution.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Utilisez les packages `SecurityToken` et `SecurityToken.Model`.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Obtenez des informations d'identification temporaires auprès de `AmazonSimpleTokenService` et créez un objet `ClusterDaxClient`. N'oubliez pas de *`user input placeholder`* remplacer chacune par les valeurs correctes pour vos comptes.

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;
```

```
var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

Cette section vous aide à modifier votre code client DAX existant pour autoriser l'accès DAX entre comptes. Si vous n'avez pas encore de code client DAX, vous pouvez trouver des exemples de code fonctionnel sur GitHub.

1. Importez les packages de session AWS STS et.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. Obtenez des informations d'identification temporaires auprès de `AmazonSimpleTokenService` et créez un objet client DAX. N'oubliez pas de *`user input placeholder`* remplacer chacune par les valeurs correctes pour vos comptes.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
```

```
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

Python

Cette section vous aide à modifier votre code client DAX existant pour autoriser l'accès DAX entre comptes. Si vous n'avez pas encore de code client DAX, vous pouvez trouver des exemples de codes fonctionnels dans le didacticiel Python et DAX.

1. Importez `boto3`.

```
import boto3
```

2. Obtenez des informations d'identification temporaires auprès de `sts` et créez un objet `AmazonDaxClient`. N'oubliez pas de *`user input placeholder`* remplacer chacune par les valeurs correctes pour vos comptes.

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']

dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

Node.js

Cette section vous aide à modifier votre code client DAX existant pour autoriser l'accès DAX entre comptes. Si vous n'avez pas encore de code client DAX, vous pouvez trouver des exemples de codes fonctionnels dans le didacticiel Node.js et DAX. N'oubliez pas de *`user input placeholder`* remplacer chacune par les valeurs correctes pour vos comptes.

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

Guide de dimensionnement de cluster DAX

Ce guide fournit des conseils pour choisir une taille de cluster Amazon DynamoDB Accelerator (DAX) et un type de nœud appropriés pour votre application. Ces instructions vous guident dans les étapes d'estimation du trafic DAX de votre application, de sélection d'une configuration de cluster et de test de celle-ci.

Si vous avez un cluster DAX et souhaitez évaluer s'il a le nombre et la taille appropriés de nœuds, consultez Mise à l'échelle d'un cluster DAX.

Rubriques

- Présentation
- Estimation du trafic
- Test de charge

Présentation

Il est important de mettre à l'échelle votre cluster de façon appropriée pour votre charge de travail, que vous créiez un cluster ou mainteniez un cluster existant. Au fur et à mesure que le temps passe et que la charge de travail de votre application change, vous devez revoir périodiquement vos décisions de mise à l'échelle pour vous assurer qu'elles sont toujours appropriées.

Le processus suit généralement les étapes suivantes :

1. Estimation du trafic. Au cours de cette étape, vous faites des prédictions sur le volume de trafic que votre application enverra à DAX, la nature du trafic (opérations de lecture et d'écriture) et le taux d'accès attendu au cache.
2. Test de charge. Dans cette étape, vous créez un cluster et lui envoyez un trafic reflétant en miroir vos estimations de l'étape précédente. Répétez cette étape jusqu'à ce que vous trouviez une configuration de cluster appropriée.
3. Surveillance de la production. Pendant que votre application utilise DAX en production, vous devez surveiller le cluster pour vérifier en permanence qu'il est toujours mis à l'échelle correctement à mesure que votre charge de travail évolue au fil du temps.

Estimation du trafic

Trois facteurs principaux caractérisent une charge de travail DAX typique :

- Taux d'accès au cache
- Unités de capacité de lecture (RCU) par seconde
- Unités de capacité d'écriture (WCU) par seconde

Estimation du taux d'accès au cache

Si vous disposez d'un cluster DAX, vous pouvez utiliser les métriques Amazon CloudWatch `ItemCacheHits` et `ItemCacheMisses` pour déterminer le taux d'accès au cache. Le taux d'accès au cache est égal à `ItemCacheHits` / (`ItemCacheHits` + `ItemCacheMisses`). Si votre charge de travail inclut les opérations `Query` ou `Scan`, vous devez également examiner les métriques `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` et `ScanCacheMisses`. Les taux d'accès au cache varient d'une application à l'autre et sont fortement influencés par le paramètre de time-to-live (TTL) du cluster. Les taux d'accès typiques pour les applications utilisant DAX sont de 85 à 95 %.

Estimation des unités de capacité en lecture et en écriture

Si vous avez déjà des tables DynamoDB pour votre application, consultez les métriques CloudWatch`ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits`. Utilisez la statistique `Sum` et divisez par le nombre de secondes de la période.

Si vous disposez également d'un cluster DAX, n'oubliez pas que la métrique DynamoDB `ConsumedReadCapacityUnits` ne tient compte que des échecs d'accès au cache. Donc, pour avoir une idée des unités de capacité de lecture par seconde gérées par votre cluster DAX, divisez le nombre par votre taux d'échec d'accès au cache (c'est-à-dire 1 - taux d'accès au cache).

Si vous n'avez pas encore de table DynamoDB, consultez la documentation sur les unités de capacité de lecture et les unités de capacité d'écriture pour estimer votre trafic en fonction du débit de demandes estimé de votre application, des éléments consultés par demande et de la taille d'élément.

Lors de l'estimation du trafic, planifiez la croissance future et les pics prévus et imprévus pour vous assurer que votre cluster dispose d'une marge de manœuvre suffisante pour augmenter le trafic.

Test de charge

L'étape suivante après l'estimation du trafic consiste à tester la configuration du cluster sous charge.

1. Pour votre test de charge initial, nous vous recommandons de commencer avec le type de nœud `dax.r4.large`, type de nœud optimisé pour la mémoire aux performances fixes les moins coûteuses.
2. Un cluster tolérant les pannes nécessite au moins trois nœuds répartis sur trois zones de disponibilité. Dans ce cas, si une zone de disponibilité devient indisponible, le nombre effectif de zones de disponibilité est réduit d'un tiers. Pour votre test de charge initial, nous vous recommandons de commencer par un cluster à deux nœuds, qui simule l'échec d'une zone de disponibilité dans un cluster à trois nœuds.
3. Conduisez un trafic soutenu (tel qu'estimé à l'étape précédente) vers votre cluster de test pendant toute la durée du test de charge.
4. Surveillez les performances du cluster pendant l'essai de charge.

Idéalement, le profil de trafic que vous conduisez pendant le test de charge devrait être aussi similaire que possible au trafic réel de votre application. Cela inclut la répartition des opérations (par exemple, 70 % `GetItem`, 25 % `Query`, et 5 % `PutItem`), le taux de demande pour chaque opération, le nombre d'éléments consultés par demande et la répartition des tailles d'articles. Pour obtenir un taux d'accès au cache similaire au taux de succès attendu de votre application, portez une attention particulière à la distribution des clés dans votre trafic de test.

> **Note**
>
> Soyez prudent lors du test de charge des types de nœuds T2 (`dax.t2.small` et `dax.t2.medium`). Les types de nœuds T2 offrent des performances CPU extensibles qui varient au fil du temps en fonction du solde de crédit CPU du nœud. Un cluster DAX s'exécutant sur des nœuds T2 peut sembler fonctionner normalement, mais si un nœud dépasse les performances de base de son instance, le nœud dépense son solde de crédits UC accumulé. Lorsque le solde créditeur est bas, les performances sont progressivement ramenées au niveau de performance de base.

Surveillez votre cluster DAX pendant le test de charge afin de déterminer si le type de nœud que vous utilisez pour le test de charge est le type de nœud qui vous convient. En outre, lors d'un test de charge, vous devez surveiller le taux de demande et le taux d'accès au cache afin de vous assurer que votre infrastructure de test entraîne réellement la quantité de trafic que vous souhaitez.

Vous devez faire attention à la consommation d'octets réseau pour le type d'instance de cluster que vous avez choisi. Le dépassement de la bande passante de référence disponible pour une instance

Amazon EC2 indique que votre cluster risque de ne pas supporter la charge de travail de votre application et qu'il doit être mis à l'échelle.

Si le test de charge indique que la configuration de cluster choisie ne peut pas supporter la charge de travail de votre application, vous avez tout intérêt à passer à un type de nœud de plus grande taille, surtout si vous constatez une utilisation d'UC élevée sur le nœud primaire du cluster, des taux d'expulsion élevés ou une utilisation de mémoire cache élevée. Si les taux d'accès sont constamment élevés et que le rapport entre le trafic lecture-écriture est élevé, vous pouvez envisager d'ajouter d'autres nœuds à votre cluster. Reportez-vous à la section Mise à l'échelle d'un cluster DAX pour savoir quand utiliser un type de nœud plus grand (mise à l'échelle verticale) ou ajouter d'autres nœuds (mise à l'échelle horizontale).

Vous devez répéter votre test de charge après avoir apporté des modifications à la configuration de votre cluster.

Modélisation des données pour les tables DynamoDB

Avant de nous plonger dans la modélisation de données, il est important de comprendre certains principes fondamentaux de DynamoDB. DynamoDB est une base de données NoSQL de type clé-valeur, ce qui procure un certaine flexibilité dans les schémas. L'ensemble des attributs de données, hormis les attributs de clé de chaque élément, peut être uniforme ou discret. Le schéma de clé DynamoDB se présente soit sous la forme d'une clé primaire simple constituée d'une clé de partition qui identifie de manière unique un élément, soit sous la forme d'une clé primaire composite constituée d'une clé de partition et d'une clé de tri qui définissent de manière unique un élément. La clé de partition est hachée pour déterminer l'emplacement physique des données et les récupérer. Il est donc important de choisir un attribut qui possède une cardinalité élevée et une capacité de mise à l'échelle horizontale en guise de clé de partition pour assurer une distribution uniforme des données. Si l'attribut de clé de tri est facultatif dans le schéma de clé, le fait de disposer d'une clé de tri permet de modéliser des relations de type « un-à-plusieurs » et de créer des collections d'éléments dans DynamoDB. Les clés de tri sont également appelées clés de plage. Celles-ci permettent de trier les éléments d'une collection d'éléments et offrent également une certaine flexibilité dans les opérations basées sur une plage.

Pour en savoir plus sur le schéma de clé DynamoDB et les bonnes pratiques associées, vous pouvez consulter les pages suivantes :

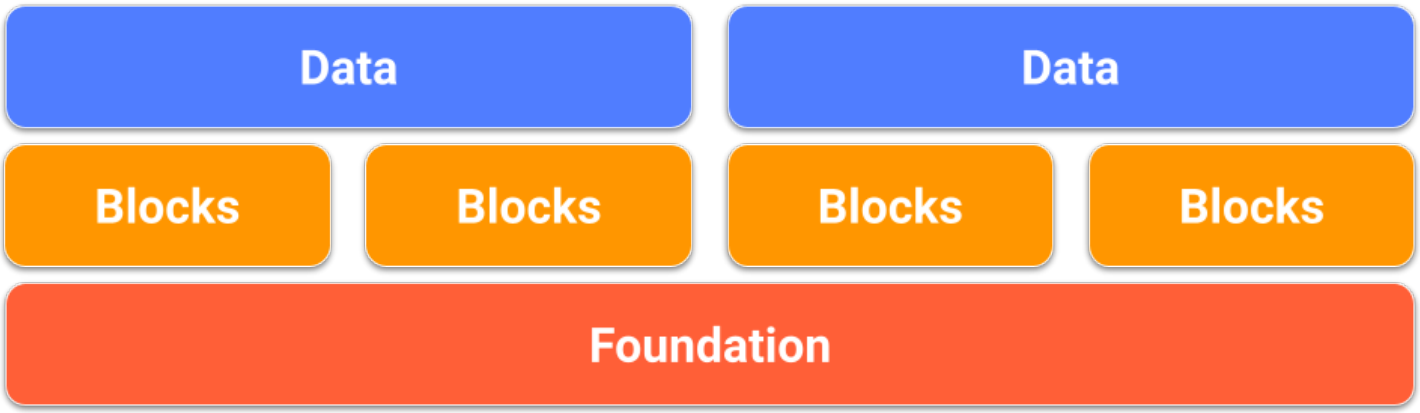
- the section called "Partitions et distribution des données dans DynamoDB"
- the section called "Création de clés de partition"
- the section called "Conception de clé de tri"
- Choix de la clé de partition DynamoDB appropriée

Il est souvent nécessaire d'utiliser des index secondaires pour prendre en charge d'autres modèles de requête dans DynamoDB. Les index secondaires sont des tables alternatives (ou « shadow ») dans lesquelles les mêmes données sont organisées via un schéma de clé différent de celui de la table de base. Dans un index secondaire local (LSI), la clé de partition est la même que celle de la table de base, ce qui permet d'avoir une autre clé de tri et ainsi de partager la capacité de la table de base. Un index secondaire global (GSI) peut avoir une clé de partition et un attribut de clé de tri différents de ceux de la table de base, ce qui signifie que la gestion du débit dans le cas d'un GSI est indépendante de la table de base.

Pour en savoir plus sur les index secondaires et les bonnes pratiques, vous pouvez consulter les pages suivantes :

- the section called “Utilisation des index”
- the section called “Index secondaires”

À présent, intéressons-nous d’un peu plus près à la modélisation de données. Le processus de conception d’un schéma flexible et hautement optimisé sur DynamoDB, ou sur n’importe quelle base de données NoSQL d’ailleurs, peut être une compétence difficile à acquérir. Ce module a pour objectif de vous aider à développer un diagramme mental pour concevoir un schéma qui vous fera passer du cas d’utilisation à la production. Nous commencerons par une introduction au choix fondamental d’un modèle, en comparant la conception à une seule table et la conception à plusieurs tables. Nous passerons ensuite en revue les différents modèles de conception (composantes de base) qui peuvent être utilisés pour obtenir divers résultats organisationnels ou de performance pour votre application. Enfin, nous verrons les packages complets de conception de schémas que nous fournissons pour différents cas d’utilisation et secteurs d’activité.



Rubriques

- Collections d'articles : comment modéliser des relations un-à-plusieurs dans DynamoDB
- Principes de base de la modélisation des données dans DynamoDB
- Composantes de base de la modélisation des données dans DynamoDB
- Packages de conception de schémas de modélisation de données dans DynamoDB
- Bonnes pratiques de modélisation des données relationnelles dans DynamoDB

Collections d'articles : comment modéliser des relations un-à-plusieurs dans DynamoDB

Dans DynamoDB, une collection d'éléments est un groupe d'éléments ayant la même valeur de clé de partition, ce qui signifie que les éléments sont liés. Les collections d'articles constituent le mécanisme principal permettant de modéliser des relations un-à-plusieurs dans DynamoDB. Les collections d'articles peuvent uniquement exister sur des tables ou des index configurés pour utiliser une clé primaire composite.

> **Note**
>
> Les collections d'éléments peuvent exister dans une table de base ou un index secondaire. Pour plus d'informations sur la façon dont les collections d'articles interagissent avec les index, voir Collections d'articles dans les index secondaires locaux.

Considérez le tableau suivant montrant trois utilisateurs différents et leurs inventaires en jeu :

Primary key		Attributes		
Partition key: PK	**Sort key: SK**			
account1234	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	inventory::weapons	data		
		{"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320}]}		
	login-data	pw	state	last-login
		d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	Active	1649276737
account1387	info	data		
		{"email" : "bot123@gmail.com"}		
	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	login-data	pw	state	last-login
		k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	Banned	1649456737
account1138	info	data		
		{"email" : "luh-3417@gmail.com" }		
	login-data	pw	state	last-login
		88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

Pour certains éléments de chaque collection, la clé de tri est une concaténation composée d'informations utilisées pour regrouper des données, telles que `inventory::armor`, `inventory::weapon` ou `info`. Chaque collection d'articles peut comporter une combinaison

différente de ces attributs comme clé de tri. L'utilisateur `account1234` est doté d'un élément `inventory::weapons`, mais ce n'est pas le cas de l'utilisateur `account1387` (parce qu'il n'en a pas encore trouvé). L'utilisateur `account1138` n'utilise que deux éléments pour sa clé de tri (puisqu'il n'a pas encore d'inventaire), tandis que les autres utilisateurs en utilisent trois.

DynamoDB vous permet de récupérer sélectivement des éléments de ces collections d'articles pour effectuer les opérations suivantes :

- Récupérer tous les éléments d'un utilisateur particulier
- Récupérer un seul élément d'un utilisateur particulier
- Récupérer tous les éléments d'un type spécifique appartenant à un utilisateur particulier

Accélérer les requêtes en organisant vos données avec des collections d'articles

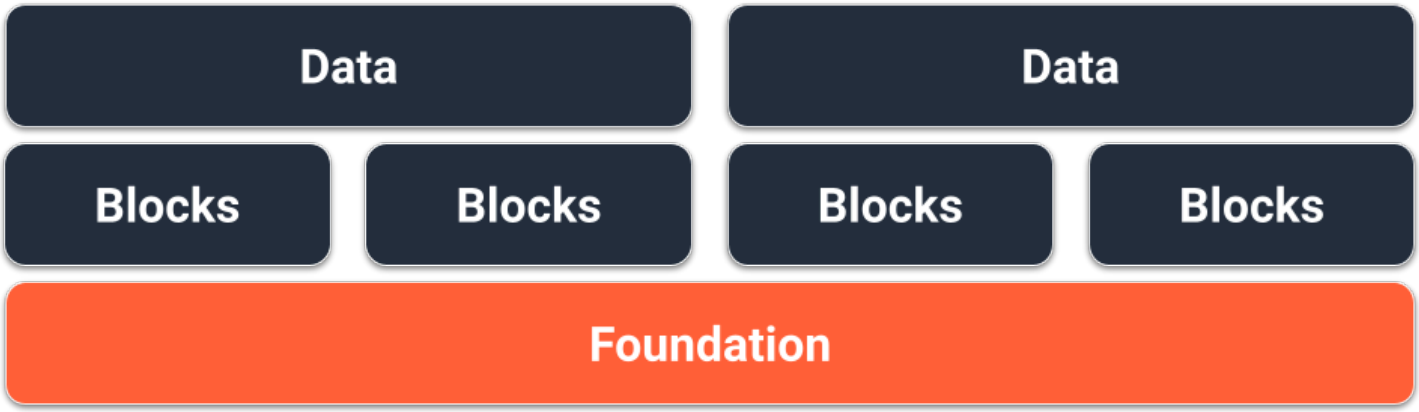
Dans cet exemple, chacun des éléments de ces trois collections d'objets représente un joueur et le modèle de données que nous avons choisi, en fonction des modèles d'accès du jeu et du joueur. De quelles données le jeu a-t-il besoin ? Quand en a-t-il besoin ? À quelle fréquence en a-t-il besoin ? Quel est le coût de ce procédé ? Ces décisions de modélisation des données ont été prises en fonction des réponses à ces questions.

Dans ce jeu, il y a une page différente présentée au joueur pour son inventaire des armes et une autre page pour les armures. Lorsque le joueur ouvre son inventaire, les armes sont affichées en premier car nous voulons que cette page se charge extrêmement rapidement, tandis que les pages d'inventaire suivantes peuvent être chargées par la suite. Étant donné que chacun de ces types d'objets peut être assez volumineux lorsque le joueur acquiert d'autres objets dans le jeu, nous avons décidé que chaque page d'inventaire serait son propre objet dans la collection d'objets du joueur dans la base de données.

La section suivante explique comment interagir avec les collections d'articles via l'opération `Query`.

Principes de base de la modélisation des données dans DynamoDB

Cette section s’intéresse à la couche de base en examinant les deux types de conception de table : à une seule table ou à plusieurs tables.



Principe de base de la conception à une seule table

Notre premier choix pour la base de notre schéma DynamoDB se porte sur la conception à une seule table. La conception à une seule table est un modèle qui vous permet de stocker plusieurs types (entités) de données dans une seule table DynamoDB. Elle vise à optimiser les modèles d'accès aux données, à améliorer les performances et à réduire les coûts en évitant de gérer plusieurs tables et les relations complexes entre elles. Cela est possible, car DynamoDB stocke les éléments dotés de la même clé de partition (appelée collection d'éléments) sur les mêmes partitions les uns que les autres. Dans cette conception, différents types de données sont stockés sous forme d'éléments dans la même table et chaque élément est identifié par une clé de tri unique.

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Avantages

- Localité des données pour prendre en charge les requêtes relatives à plusieurs types d'entités lors d'un seul appel de base de données
- Réduit les coûts financiers globaux et les coûts de latence liés aux lectures :
  - Une seule requête pour deux éléments d'un volume total inférieur à 4 ko équivaut à 0,5 RCU finalement cohérente
  - Deux requêtes pour deux éléments d'un volume total inférieur à 4 ko équivalent à 1 RCU finalement cohérente (0,5 RCU chacune)
  - La durée nécessaire pour renvoyer deux appels de base de données distincts sera en moyenne supérieure à celle d'un seul appel
- Réduit le nombre de tables à gérer :
  - Il n'est pas nécessaire de conserver les autorisations entre plusieurs rôles ou politiques IAM
  - La gestion de la capacité de la table est calculée en moyenne pour toutes les entités, ce qui se traduit généralement par un modèle de consommation plus prévisible

  - La surveillance nécessite moins d'alarmes
  - Les clés de chiffrement gérées par le client ne doivent subir une rotation que sur une table
- Facilite le trafic vers la table :
  - En agrégeant plusieurs modèles d'utilisation dans la même table, l'utilisation globale a tendance à être plus fluide (de la même manière que la performance d'un indice boursier a tendance à être plus fluide que celle d'une action individuelle), ce qui fonctionne mieux pour obtenir une utilisation plus élevée avec les tables en mode provisionné

Inconvénients

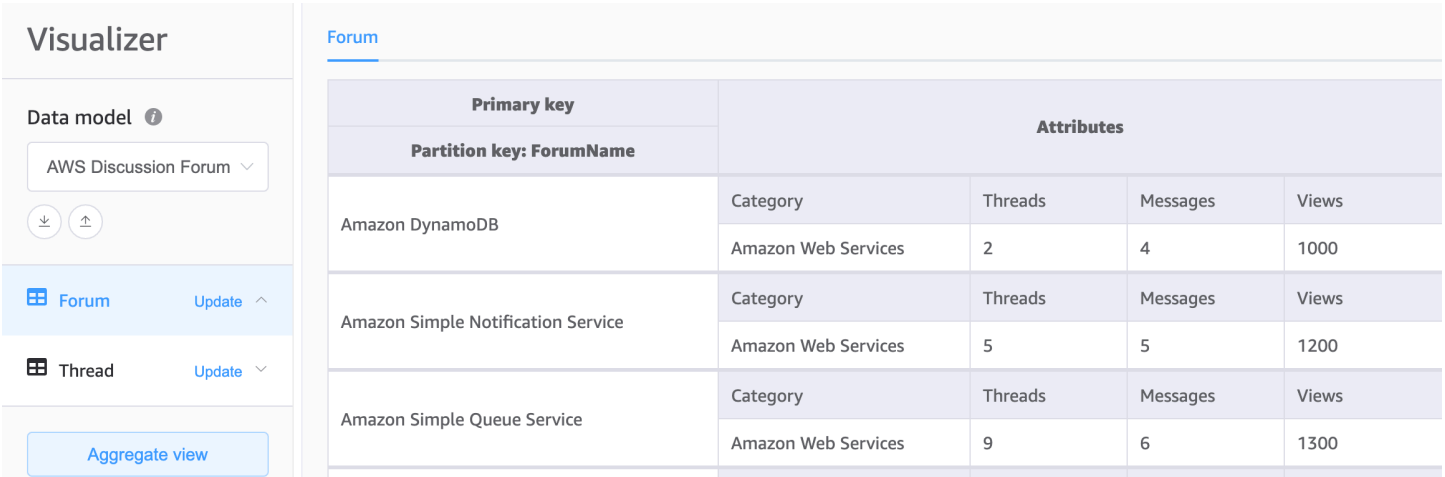
- La courbe d'apprentissage peut être ardue en raison d'une conception paradoxale par rapport aux bases de données relationnelles
- Les exigences en matière de données doivent être cohérentes pour tous les types d'entités
  - Pour les sauvegardes, c'est tout ou rien. Par conséquent, si certaines données ne sont pas essentielles à votre mission, pensez à les conserver dans une table séparée
  - Le chiffrement des tables est partagé entre tous les éléments. Pour les applications multilocataires avec des exigences de chiffrement de locataire individuel, un chiffrement côté client est requis
  - Les tables contenant à la fois des données historiques et des données opérationnelles ne tireront pas autant d'avantages de l'activation de la classe de stockage Accès peu fréquent. Pour plus d'informations, consultez Classes de tables DynamoDB.
- Toutes les données modifiées seront transmises à DynamoDB Streams, même si seul un sous-ensemble d'entités doit être traité.
  - Grâce aux filtres d'événements Lambda, cela n'affectera pas votre facture lorsque vous utiliserez Lambda, mais cela représentera un coût supplémentaire lorsque vous utiliserez Kinesis Consumer Library
- Lors de l'utilisation de GraphQL, la conception à une seule table sera plus difficile à mettre en œuvre.
- Lorsque vous utilisez des clients SDK de niveau supérieur tels que `DynamoDBMapper` de Java ou le client amélioré, le traitement des résultats peut s'avérer plus difficile, car les éléments d'une même réponse peuvent être associés à différentes classes

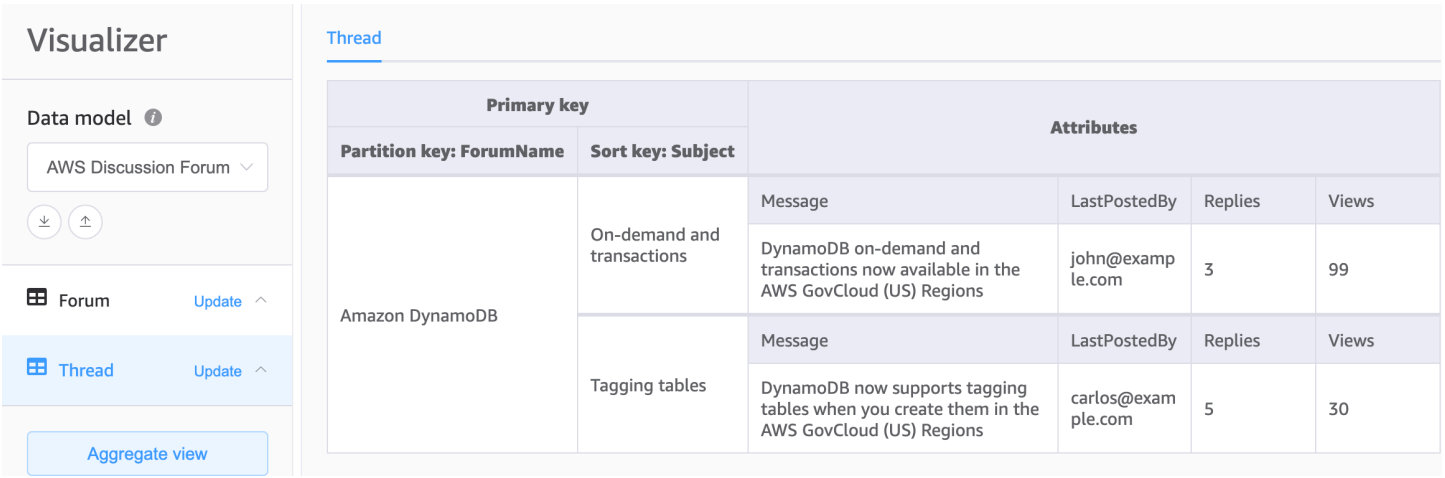
Quand l'utiliser

La conception à une seule table est indiquée pour les applications qui interrogent fréquemment plusieurs types d'entités ensemble ou qui ont besoin de maintenir des relations entre différents types de données. Elle est particulièrement efficace lorsque vos modèles d'accès tirent parti de la localisation des données et lorsque vous souhaitez minimiser les frais liés à la gestion de plusieurs tables.

Principe de base de la conception à plusieurs tables

Notre second choix pour la base de notre schéma DynamoDB se porte sur la conception à plusieurs tables. La conception à plusieurs tables est un modèle qui ressemble davantage à une conception de base de données traditionnelle dans laquelle vous stockez un seul type (entité) de données dans chaque table DynamoDB. Les données de chaque table seront toujours organisées par clé de partition, de sorte que les performances au sein d'un même type d'entité seront optimisées en termes de capacité de mise à l'échelle et de performance, mais les requêtes sur plusieurs tables devront être effectuées indépendamment.





Avantages

- Plus simple à concevoir pour qui n'a pas l'habitude d'utiliser la conception à une seule table.
- Mise en œuvre simplifiée des résolveurs GraphQL grâce au mappage de chaque résolveur à une seule entité (table)
- Permet de répondre à des exigences de données uniques pour différents types d'entités :
  - Possibilité d'effectuer des sauvegardes pour les tables individuelles qui sont essentielles à la mission
  - Le chiffrement des tables peut être géré pour chaque table. Pour les applications multilocataires ayant des exigences de chiffrement de locataire individuel, les tables de locataire distinctes permettent à chaque client d'avoir sa propre clé de chiffrement
  - La classe de stockage Accès peu fréquent peut être activée uniquement sur les tables contenant des données historiques afin de tirer pleinement parti des économies sur les coûts. Pour plus d'informations, consultez Classes de tables DynamoDB.
- Chaque table disposera de son propre flux de données de modification, ce qui permettra de concevoir une fonction Lambda dédiée pour chaque type d'élément plutôt qu'un seul processeur monolithique.

Inconvénients

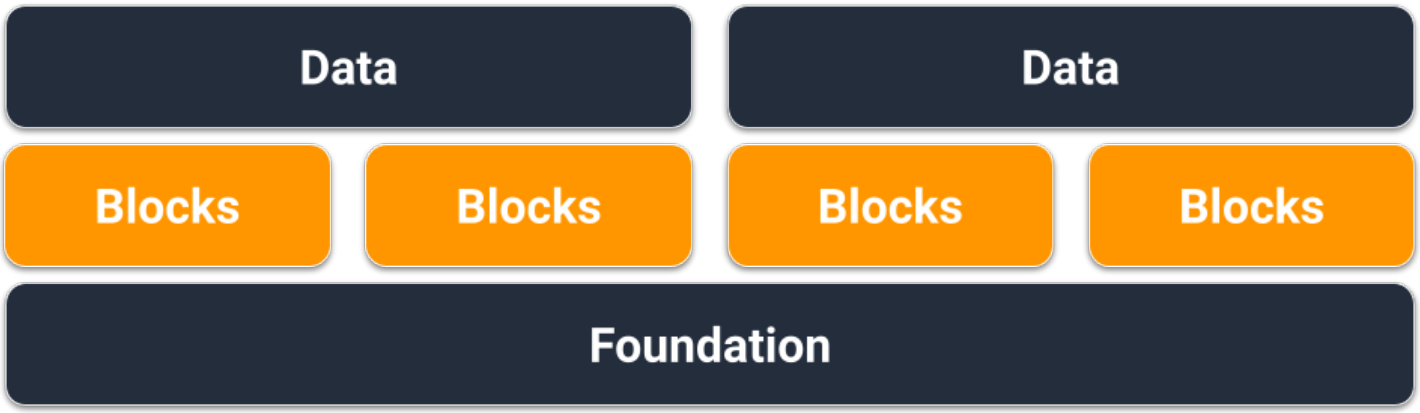
- Pour les modèles d'accès qui exigent des données sur plusieurs tables, plusieurs lectures seront exigées à partir de DynamoDB et les données devront peut-être être traitées/jointes sur le code client.
- Les opérations et la surveillance de plusieurs tables exigent davantage d'alarmes CloudWatch et chaque table doit être mise à l'échelle indépendamment.
- Les autorisations de chaque table devront être gérées séparément. L'ajout de tables à l'avenir exigera de modifier tous les rôles ou politiques IAM nécessaires.

Quand l'utiliser

Si les modèles d'accès de votre application n'ont pas besoin d'interroger simultanément plusieurs entités ou tables, la conception à plusieurs tables est une approche appropriée et suffisante.

Composantes de base de la modélisation des données dans DynamoDB

Cette section s'intéresse à la couche des composants principaux en vous présentant des modèles de conception que vous pouvez utiliser dans votre application.



Rubriques

- Composante de base Clé de tri composite
- Composante de base Multilocataire
- Composante de base Index fragmenté
- Composante de base Durée de vie
- Composante de base Durée de vie pour l'archivage
- Composante de base Partitionnement vertical
- Composante de base Partitionnement d'écriture

Composante de base Clé de tri composite

Quand on pense à NoSQL, on peut également le considérer comme non relationnel. En fin de compte, il n'y a aucune raison pour que les relations ne puissent pas être placées dans un schéma DynamoDB. Elles ont simplement un aspect différent de celui des bases de données relationnelles et de leurs clés étrangères. L'un des modèles les plus importants que nous pouvons utiliser pour développer une hiérarchie logique de nos données dans DynamoDB est la clé de tri composite. Pour en concevoir une, il convient généralement de séparer chaque couche de la hiérarchie (couche parent > couche enfant > couche petit-enfant) par un hashtag. Par exemple, `PARENT#CHILD#GRANDCHILD#ETC`.

Primary key	
Partition key: PK	**Sort key: SK**
UserID	CART#ACTIVE#Apples
	CART#ACTIVE#Bananas
	CART#SAVED#Oranges
	CART#SAVED#Pears
	WISH#VEGGIES#Carrots

Alors que dans DynamoDB, une clé de partition exige toujours la valeur exacte pour interroger les données, nous pouvons appliquer une condition partielle à la clé de tri de gauche à droite, comme pour traverser une arborescence binaire.

L'exemple ci-dessus s'applique à une boutique de e-commerce où le panier doit être conservé entre les sessions utilisateur. Chaque fois que l'utilisateur se connecte, il est possible qu'il souhaite voir l'intégralité de son panier, y compris les articles enregistrés pour plus tard. Mais lorsqu'il passe au paiement, seuls les articles du panier actif doivent être chargés pour l'achat. Étant donné que ces deux `KeyConditions` demandent explicitement les clés de tri CART, les données supplémentaires de la liste de souhaits sont simplement ignorées par DynamoDB au moment de la lecture. Bien que les articles enregistrés et actifs fassent partie du même panier, nous devons les traiter différemment selon les parties de l'application. L'application d'une expression `KeyCondition` au préfixe de la clé de tri est donc le moyen le plus optimisé de récupérer uniquement les données nécessaires pour chaque partie de l'application.

Fonctionnalités principales de cette composante de base

- Les articles associés sont stockés localement les uns par rapport aux autres pour accéder efficacement aux données.
- À l'aide d'`KeyCondition`expressions, les sous-ensembles de la hiérarchie peuvent être récupérés de manière sélective, ce qui signifie qu'il n'y a aucun gaspillage RCUs
- Les différentes parties de l'application peuvent stocker leurs articles sous un préfixe spécifique pour éviter le remplacement des articles ou les conflits d'écriture.

Composante de base Multilocataire

De nombreux clients utilisent DynamoDB pour héberger les données de leurs applications multilocataires. Pour ces scénarios, nous souhaitons concevoir le schéma de manière à conserver toutes les données d'un seul locataire dans sa propre partition logique de la table. On s'appuie sur le concept de collection d'éléments, qui désigne tous les éléments d'une table DynamoDB ayant la même clé de partition. Pour plus d'informations sur la manière dont DynamoDB aborde la multilocation, consultez Multitenancy on DynamoDB.

Primary key		Attributes
Partition key: PK	**Sort key: SK**	
UserOne	PhotoID1	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID2	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserTwo	PhotoID3	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID4	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserThree	PhotoID5	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]

Cet exemple s'applique à un site d'hébergement de photos qui compte potentiellement des milliers d'utilisateurs. Dans un premier temps, chaque utilisateur charge uniquement des photos sur son propre profil, mais par défaut, nous n'autorisons pas un utilisateur à voir les photos d'un autre utilisateur. Dans l'idéal, un niveau d'isolation supplémentaire devrait être ajouté à l'autorisation d'appel de chaque utilisateur à votre API afin de garantir qu'il ne demande que des données issues de sa propre partition. Au niveau du schéma, les clés de partition uniques sont suffisantes.

Fonctionnalités principales de cette composante de base

- La quantité de données lues par un utilisateur ou un locataire ne peut être qu'égale à la quantité totale d'éléments dans sa partition.
- La suppression des données d'un locataire en raison de la fermeture de son compte ou d'une demande de conformité peut être effectuée avec tact et à moindre coût. Il suffit

d'exécuter une requête où la clé de partition est égale à l'ID du locataire, puis d'exécuter une opération `DeleteItem` pour chaque clé primaire renvoyée.

> **Note**
>
> Conçu dans un souci de mutualisation, vous pouvez utiliser différents fournisseurs de clés de chiffrement sur une même table pour isoler les données en toute sécurité. AWS Database Encryption SDK pour Amazon DynamoDB vous permet d'inclure le chiffrement côté client dans vos charges de travail DynamoDB. Vous pouvez effectuer un chiffrement au niveau des attributs, ce qui vous permet de chiffrer des valeurs d'attributs spécifiques avant de les stocker dans votre table DynamoDB et de rechercher des attributs chiffrés sans déchiffrer l'intégralité de la base de données au préalable.

Composante de base Index fragmenté

Parfois, un modèle d'accès exige de rechercher des éléments qui correspondent à un élément rare ou à un élément qui reçoit un statut (ce qui nécessite une réponse ayant fait l'objet d'une remontée). Plutôt que d'effectuer régulièrement des recherches sur l'ensemble du jeu de données pour ces éléments, nous pouvons tirer parti du fait que peu de données sont chargées dans les index secondaires globaux (GSI). Cela signifie que seuls les éléments de la table de base, dont les attributs sont définis dans l'index, seront répliqués dans l'index.

Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

Cet exemple illustre un cas d'utilisation IOT où chaque appareil sur le terrain communique régulièrement un état. Pour la plupart des signalements, on s'attend à ce que l'appareil indique que tout va bien, mais il peut arriver qu'il y ait un problème et qu'il faille le faire remonter à un réparateur. Pour les signalements nécessitant une remontée, l'attribut `EscalatedTo` est ajouté à l'élément, mais il n'est pas présent dans le cas contraire. Dans cet exemple, le GSI est partitionné par `EscalatedTo` et étant donné qu'il apporte les clés de la table de base, on peut voir quel DeviceID a signalé le problème et à quel moment.

Bien que les lectures soient moins coûteuses que les écritures dans DynamoDB, les index fragmentés constituent un outil très puissant pour les cas d'utilisation où les instances d'un type spécifique d'élément sont rares, mais où les lectures pour les trouver sont courantes.

Fonctionnalités principales de cette composante de base

- Les coûts d'écriture et de stockage pour le GSI fragmenté ne s'appliquent qu'aux éléments qui correspondent au modèle clé, de sorte que le coût du GSI peut être nettement inférieur à GSIs celui d'autres modèles où tous les éléments sont répliqués
- Une clé de tri composite peut toujours être utilisée pour affiner davantage les éléments qui correspondent à la requête souhaitée. Par exemple, un horodatage peut être utilisé pour la clé de tri afin d'afficher uniquement les problèmes signalés au cours des X dernières minutes (`SK > 5 minutes ago, ScanIndexForward: False`).

Composante de base Durée de vie

La plupart des données ont une certaine durée pendant laquelle on peut considérer qu'il vaut la peine de les conserver dans un entrepôt de données principal. Pour faciliter le vieillissement des données issues de DynamoDB, il existe une fonctionnalité appelée Durée de vie (TTL). La fonctionnalité TTL vous permet de définir un attribut spécifique au niveau de la table qui surveille les éléments dotés d'un horodatage époque (dans le passé). Cela vous permet de supprimer gratuitement les enregistrements expirés de la table.

> **Note**
>
> Si vous utilisez la version 2019.11.21 (actuelle) des tables globales et que vous utilisez également la fonctionnalité Durée de vie, DynamoDB réplique les suppressions de TTL sur toutes les tables de réplica. La suppression de TTL initiale ne consomme pas de capacité d'écriture dans la région dans laquelle l'expiration de TTL a lieu. Toutefois, la suppression de TTL répliquée dans la ou les tables de réplicas consomme de la capacité d'écriture répliquée dans chacune des régions de réplica et des frais s'appliquent.

Primary key		Attributes	
Partition key: PK	**Sort key: MessageTimestamp**		
UserID	2030-06-30T12:12:12	TTL	Message
		1909570332	Hello
	2030-06-30T12:17:22	TTL	Message
		1909570647	DynamoDB
	2030-06-30T12:22:27	TTL	Message
		1909570947	TTL

Cet exemple illustre une application conçue pour permettre à un utilisateur de créer des messages à durée de vie limitée. Lorsqu'un message est créé dans DynamoDB, l'attribut TTL est défini sur une date postérieure de sept jours par le code de l'application. Dans environ sept jours, DynamoDB constatera que l'horodatage epoch de ces éléments sera passé et les supprimera.

Les suppressions effectuées par TTL étant gratuites, il est fortement recommandé d'utiliser cette fonctionnalité pour supprimer les données historiques de la table. Cela réduira la facture mensuelle de stockage global et probablement les coûts de lecture des utilisateurs, car leurs requêtes nécessiteront de récupérer moins de données. Bien que la TTL soit activée au niveau de la table, c'est à vous de décider pour quels éléments ou entités vous souhaitez créer un attribut TTL et jusqu'à quelle date vous souhaitez définir l'horodatage epoch.

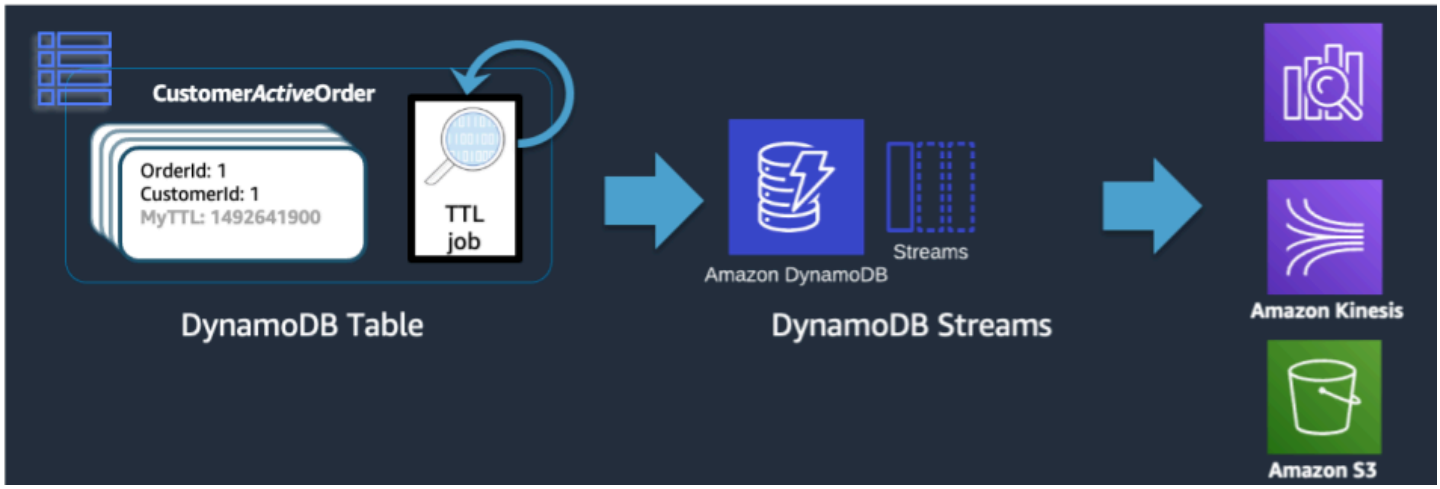
Fonctionnalités principales de cette composante de base

- Les suppressions TTL sont exécutées en arrière-plan, sans incidence sur les performances de votre table.

- La TTL est un processus asynchrone qui s'exécute environ toutes les 6 heures, mais la suppression d'un enregistrement expiré peut prendre plus de 48 heures.
  - Ne vous fiez pas aux suppressions TTL pour des cas d'utilisation tels que le verrouillage des enregistrements ou la gestion de l'état si des données obsolètes doivent être nettoyées en moins de 48 heures.
- Vous pouvez attribuer un nom valide à l'attribut TTL, mais la valeur doit être de type numérique.

Composante de base Durée de vie pour l'archivage

Bien que la TTL soit un outil efficace pour supprimer des données anciennes de DynamoDB, de nombreux cas d'utilisation exigent de conserver une archive des données pendant une période plus longue que celle de l'entrepôt de données principal. Dans ce cas, nous pouvons tirer parti de la suppression temporisée des enregistrements par TTL pour transférer les enregistrements expirés vers un entrepôt de données à long terme.



Lorsqu'une suppression TTL est effectuée par DynamoDB, elle est tout de même transférée au flux DynamoDB en tant qu'événement `Delete`. Cependant, lorsque la TTL DynamoDB effectue la suppression, il existe un attribut sur l'enregistrement du flux de `principal:dynamodb`. En utilisant un abonné Lambda au DynamoDB Stream, nous pouvons appliquer un filtre d'événements uniquement pour l'attribut principal DynamoDB et savoir que tous les enregistrements correspondant à ce filtre doivent être transférés vers un magasin d'archives tel qu'Amazon Glacier.

Fonctionnalités principales de cette composante de base

- Une fois que les lectures à faible latence de DynamoDB ne sont plus nécessaires pour les éléments historiques, leur migration vers un service de stockage plus froid tel qu'Amazon Glacier

peut réduire les coûts de stockage de manière significative tout en répondant aux exigences de conformité des données de votre cas d'utilisation

- Si les données sont conservées dans Amazon S3, des outils d’analyse rentables tels qu’Amazon Athena ou Redshift Spectrum peuvent être utilisés pour effectuer une analyse historique des données.

Composante de base Partitionnement vertical

Les utilisateurs qui connaissent la base de données de modèles de documents seront familiers avec l’idée de stocker toutes les données associées dans un seul document JSON. DynamoDB prend en charge les types de données JSON, mais ne prend pas en charge l’exécution des `KeyConditions` sur les données JSON imbriquées. Puisque ce `KeyConditions` sont eux qui déterminent la quantité de données lues sur le disque et la consommation effective RCUs d'une requête, cela peut entraîner des inefficiences à grande échelle. Pour mieux optimiser les écritures et les lectures de DynamoDB, nous vous recommandons de diviser les entités individuelles du document en éléments DynamoDB individuels. Cette méthode s’appelle également le partitionnement vertical.

```
{
        ”UserProfile” : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store” : {
                ”store_id”: ”STOREUID”,
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID” : {
                ”ProductA”: ”SKU_A”,
                ”ProductB”: ”SKU_B”,
                "DateOrdered":”2018-09-28"}
}
```

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Le partitionnement vertical, comme indiqué ci-dessus, est un exemple clé de conception à une seule table en action, mais il peut également être mis en œuvre sur plusieurs tables si vous le souhaitez. Étant donné que DynamoDB facture les écritures par incréments de 1 ko, vous devez idéalement partitionner le document de manière à obtenir des éléments inférieurs à 1 ko.

Fonctionnalités principales de cette composante de base

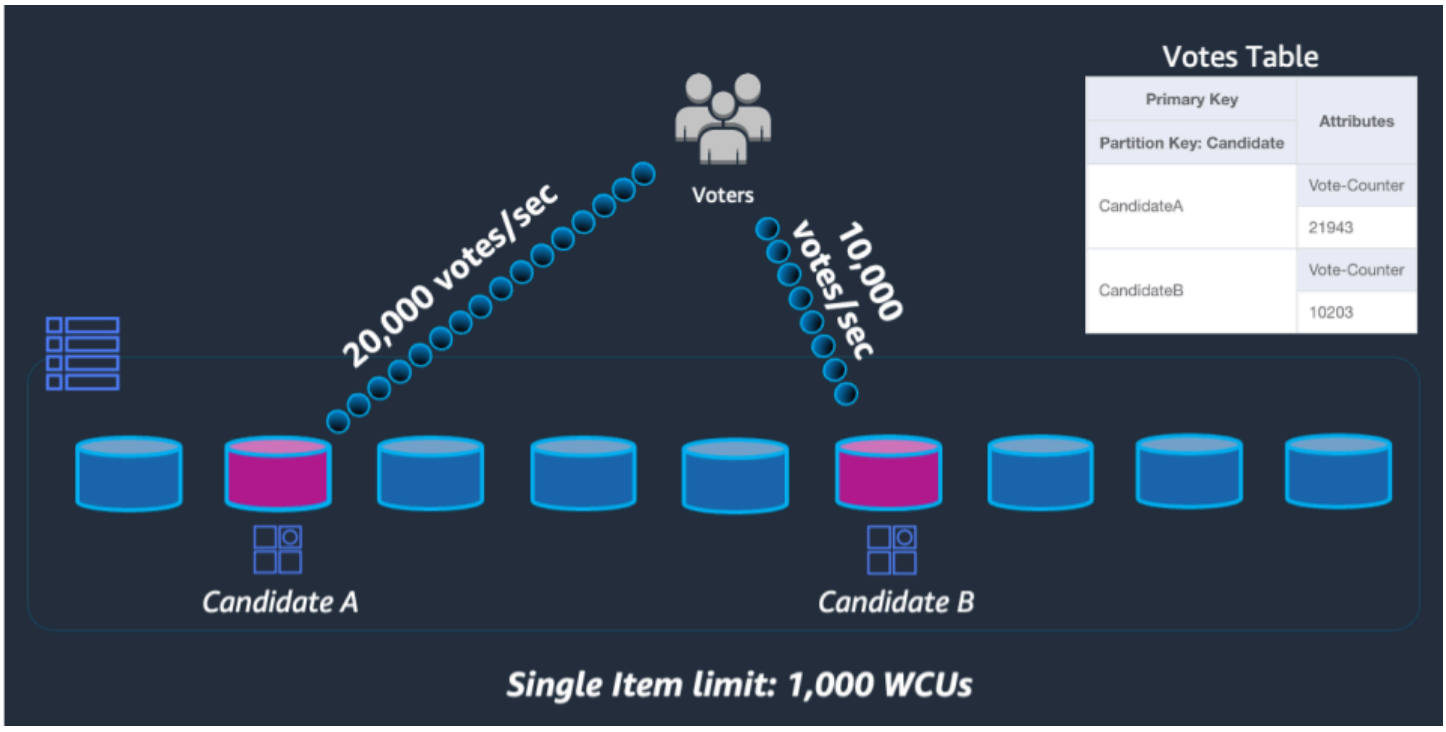
- Une hiérarchie des relations entre les données est maintenue via des préfixes de clé de tri, de sorte que la structure unique du document puisse être reconstruite côté client si nécessaire.
- Des composants uniques de la structure de données peuvent être mis à jour indépendamment, de sorte que les mises à jour de petits éléments ne représentant qu'une seule WCU.
- À l'aide de la clé de tri `BeginsWith`, l'application peut récupérer des données similaires dans une seule requête, en agrégeant les coûts de lecture pour réduire le coût total/la latence.
- Étant donné que les documents volumineux peuvent facilement dépasser la limite de 400 ko pour chaque élément dans DynamoDB, le partitionnement vertical permet de contourner cette limite.

Composante de base Partitionnement d'écriture

L'une des rares limites strictes mises en place par DynamoDB est la limitation du débit qu'une seule partition physique peut maintenir par seconde (pas nécessairement une clé de partition unique). Ces limites sont actuellement les suivantes :

- 1 000 WCU (ou 1 000 éléments écrits par seconde <= 1 ko) et 3 000 RCU (ou 3 000 lectures par seconde <=4 ko) fortement cohérentes ou
- 6 000 lectures par seconde <=4 ko finalement cohérentes

Si les demandes adressées à la table dépassent l'une de ces limites, une erreur est renvoyée au SDK client de `ThroughputExceededException`, ce que l'on appelle plus communément la limitation. Les cas d'utilisation exigeant des opérations de lecture au-delà de cette limite seront généralement mieux servis en plaçant un cache de lecture devant DynamoDB. Les opérations d'écriture, elles, exigent une conception au niveau du schéma connue sous le nom de partitionnement d'écriture.



Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53

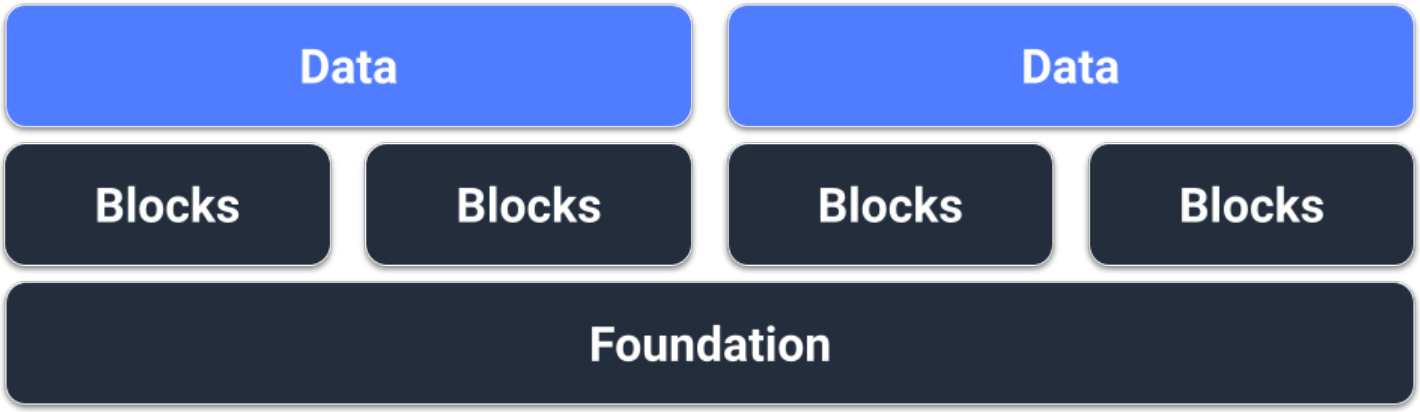
Pour résoudre ce problème, nous allons ajouter un entier aléatoire à la fin de la clé de partition pour chaque concurrent dans le code `UpdateItem` de l'application. La plage du générateur d'entiers aléatoires devra avoir une limite supérieure égale ou supérieure au nombre attendu d'écritures par seconde pour un concurrent donné divisé par 1 000. Pour obtenir 20 000 votes par seconde, l'expression ressemblerait à rand(0,19). Maintenant que les données sont stockées sous des partitions logiques distinctes, elles doivent être recombinées au moment de la lecture. Comme le total des votes n'a pas besoin d'être en temps réel, une fonction Lambda programmée pour lire toutes les partitions de vote toutes les X minutes pourrait effectuer une agrégation occasionnelle pour chaque concurrent et la réécrire dans un enregistrement du total des votes unique pour les lectures en direct.

Fonctionnalités principales de cette composante de base

- Pour les cas d'utilisation impliquant un débit d'écriture extrêmement élevé pour une clé de partition donnée qui ne peut être évité, les opérations d'écriture peuvent être réparties artificiellement sur plusieurs partitions DynamoDB.
- GSIs avec une faible cardinalité, une clé de partition devrait également utiliser ce modèle, car la régulation d'un GSI exercera une contre-pression sur les opérations d'écriture sur la table de base.

Packages de conception de schémas de modélisation de données dans DynamoDB

Découvrez les packages de conception de schémas de modélisation des données pour DynamoDB, notamment les cas d'utilisation, les modèles d'accès et les conceptions de schéma finales pour les réseaux sociaux, les profils de jeu, la gestion des plaintes, les paiements récurrents, le statut des appareils et les boutiques en ligne.



Prérequis

Avant d'essayer de concevoir notre schéma pour DynamoDB, nous devons d'abord recueillir certaines données prérequises concernant le cas d'utilisation que le schéma doit prendre en charge. Contrairement aux bases de données relationnelles, DynamoDB est partitionné par défaut, ce qui signifie que les données seront stockées sur plusieurs serveurs en arrière-plan. Il est donc important de concevoir en fonction de la localisation des données. Nous devrons établir la liste suivante pour chaque conception de schéma :

- Liste des entités (diagramme ER)
- Volumes et débit estimés pour chaque entité
- Modèles d'accès qui doivent être pris en charge (requêtes et écritures)
- Exigences en matière de conservation des données

Rubriques

- Conception de schéma de réseau social dans DynamoDB
- Conception de schéma de profil de jeu dans DynamoDB
- Conception du schéma d'un système de gestion des réclamations dans DynamoDB
- Conception d'un schéma de paiements récurrents dans DynamoDB
- Surveillance des mises à jour du statut d'un appareil dans DynamoDB
- Utilisation de DynamoDB comme magasin de données pour un magasin en ligne

Conception de schéma de réseau social dans DynamoDB

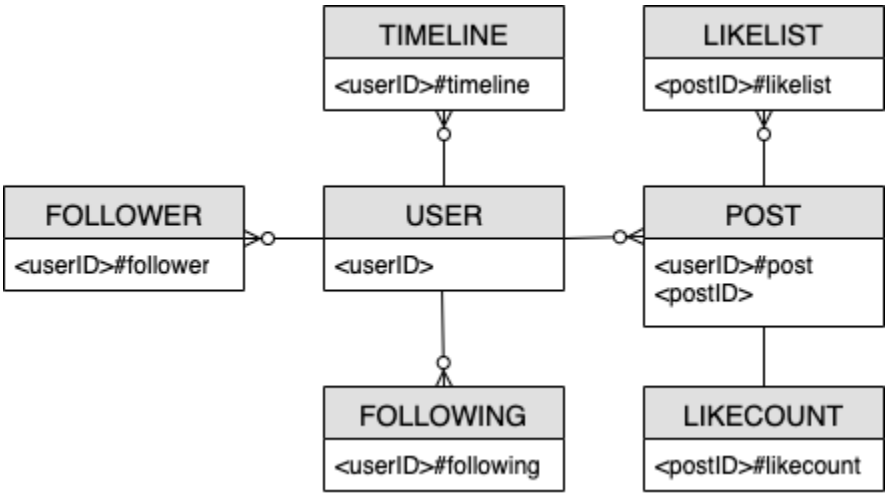
Cas d'utilisation métier de réseau social

Ce cas d'utilisation décrit l'utilisation de DynamoDB en tant que réseau social. Un réseau social est un service en ligne qui permet à différents utilisateurs d'interagir entre eux. Le réseau social que nous allons concevoir permettra à l'utilisateur de consulter une frise chronologique comprenant ses publications, ses abonnés, les personnes qu'il suit et les publications écrites par les personnes qu'il suit. Les modèles d'accès de cette conception de schéma sont les suivants :

- Obtenir des informations utilisateur pour un ID utilisateur donné
- Obtenir la liste des abonnés pour un ID utilisateur donné
- Obtenir la liste des personnes suivies pour un ID utilisateur donné
- Obtenir la liste des publications pour un ID utilisateur donné
- Obtenir la liste des utilisateurs qui aiment la publication pour un postID donné
- Obtenir le nombre de « J'aime » pour un postID donné
- Obtenir la chronologie pour un ID utilisateur donné

Diagramme des relations entre entités du réseau social

Il s'agit du diagramme des relations entre entités (ERD) que nous utiliserons pour la conception du schéma de réseau social.



Modèles d'accès de réseau social

Voici les modèles d'accès que nous allons prendre en considération pour la conception du schéma de réseau social.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Évolution de la conception du schéma de réseau social

DynamoDB étant une base de données NoSQL, elle ne vous permet pas d'effectuer une jointure, c'est-à-dire une opération qui combine des données provenant de plusieurs bases de données. Les clients qui ne connaissent pas DynamoDB peuvent appliquer les philosophies de conception du système de gestion de base de données relationnelle (SGBDR) (par exemple, créer une table pour chaque entité) à DynamoDB lorsqu'ils n'en ont pas besoin. La conception à une seule table de DynamoDB a pour objectif d'écrire des données pré-assemblées conformément au modèle d'accès de l'application, puis d'utiliser immédiatement les données sans aucun calcul supplémentaire. Pour plus d'informations, consultez Single-table vs. multi-table design in DynamoDB.

Voyons maintenant comment nous allons faire évoluer la conception de notre schéma pour prendre en compte tous les modèles d'accès.

Étape 1 : Traitement du modèle d'accès 1 (**`getUserInfoByUserID`**)

Pour obtenir les informations d'un utilisateur donné, nous allons devoir interroger (`Query`) la table de base avec la condition de clé `PK=<userID>`. L'opération de requête permet de paginer les résultats, ce qui peut être utile lorsqu'un utilisateur compte de nombreux followers. Pour plus d'informations sur Query, consultez Interrogation de tables dans DynamoDB.

Dans notre exemple, nous suivons deux types de données pour notre utilisateur : count et info. Le type de données « count » d'un utilisateur reflète le nombre d'abonnés qu'il possède, le nombre d'utilisateurs qu'il suit et le nombre de publications qu'il a créées. Le type de données « info » d'un utilisateur reflètent ses informations personnelles telles que son nom.

Ces deux types de données sont représentés par les deux éléments ci-dessous. L'élément dont la clé de tri (SK) contient « count » est plus susceptible de changer que l'élément contenant « info ». DynamoDB prend en compte la taille de l'élément telle qu'elle apparaît avant et après la mise à jour et le débit provisionné consommé reflétera la plus grande de ces tailles d'élément. Même si vous mettez à jour simplement un sous-ensemble d'attributs de l'élément, `UpdateItem` utilisera toujours la totalité du volume de débit provisionné (la plus grande des tailles d'élément « avant » et « après »). Vous pouvez obtenir les éléments en une seule opération `Query` et utiliser `UpdateItem` pour en ajouter ou en soustraire des attributs numériques existants.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...

Étape 2 : Traitement du modèle d'accès 2 (**`getFollowerListByUserID`**)

Pour obtenir la liste des utilisateurs qui suivent un utilisateur donné, nous devrons interroger (`Query`) la table de base avec la condition clé `PK=<userID>#follower`.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				

Étape 3 : Traitement du modèle d'accès 3 (**`getFollowingListByUserID`**)

Pour obtenir la liste des utilisateurs qu'un utilisateur donné suit, nous devrons interroger (`Query`) la table de base avec la condition clé `PK=<userID>#following`. Vous pouvez ensuite utiliser une opération `TransactWriteItems` pour regrouper plusieurs demandes et procéder comme suit :

- Ajoutez l'utilisateur A à la liste d'abonnés de l'utilisateur B, puis augmentez le nombre d'abonnés de l'utilisateur B de 1.
- Ajoutez l'utilisateur B à la liste d'abonnés de l'utilisateur A, puis augmentez le nombre d'abonnés de l'utilisateur A de 1.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				

Étape 4 : Traitement du modèle d'accès 4 (**`getPostListByUserID`**)

Pour obtenir la liste des publications créées par un utilisateur donné, nous devrons interroger (`Query`) la table de base avec la condition clé `PK=<userID>#post`. Il est important de noter ici que les postID d'un utilisateur doivent être incrémentiels : la deuxième valeur postID doit être supérieure à la première valeur postID (car les utilisateurs souhaitent que leurs publications soient triées). Pour ce faire, vous pouvez générer des postID sur la base d'une valeur temporelle, comme un ULID (Universally Unique Lexicographically Sortable Identifier).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	

Étape 5 : Traitement du modèle d'accès 5 (**`getUserLikesByPostID`**)

Pour obtenir la liste des utilisateurs qui ont aimé la publication d'un utilisateur donné, nous devrons interroger (`Query`) la table de base avec la condition clé `PK=<postID>#likelist`. Cette approche est la même que celle que nous avons utilisée pour récupérer les listes d'abonnés et de personnes suivies dans le modèle d'accès 2 (`getFollowerListByUserID`) et le modèle d'accès 3 (`getFollowingListByUserID`).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				

Étape 6 : Traitement du modèle d'accès 6 (**`getLikeCountByPostID`**)

Pour obtenir le nombre de « J'aime » pour une publication donnée, nous devons effectuer une opération `GetItem` sur la table de base avec la condition clé `PK=<postID>#likecount`. Ce modèle d'accès peut entraîner des problèmes de limitation à chaque fois qu'un utilisateur ayant de nombreux abonnés (comme une célébrité, par exemple) crée une publication, car la limitation se produit lorsque le débit d'une partition dépasse 1 000 WCU par seconde. Ce problème n'est pas dû à DynamoDB, il apparaît simplement dans DynamoDB puisqu'il se trouve à la fin de la pile logicielle.

Vous devriez déterminer s'il est vraiment essentiel que tous les utilisateurs puissent consulter le nombre de « J'aime » simultanément ou si cela peut se faire progressivement au fil du temps. En général, le nombre de « J'aime » d'une publication n'a pas besoin d'être immédiatement précis à 100 %. Vous pouvez mettre en œuvre cette stratégie en plaçant une file d'attente entre votre application et DynamoDB pour que les mises à jour soient effectuées régulièrement.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			

Étape 7 : Traitement du modèle d'accès 7 (**`getTimelineByUserID`**)

Pour obtenir la chronologie d'un utilisateur donné, nous devons effectuer une opération `Query` sur la table de base avec la condition clé `PK=<userID>#timeline`. Imaginons un scénario dans lequel les abonnés d'un utilisateur doivent consulter leur publication de manière synchrone. Chaque fois qu'un utilisateur écrit une publication, sa liste d'abonnés est lue et ses userID et postID sont lentement saisis dans la clé de chronologie de tous ses abonnés. Ensuite, lorsque votre application démarre, vous pouvez lire la clé de chronologie avec l'opération `Query` et remplir l'écran de la chronologie avec une combinaison de userID et de postID en utilisant l'opération `BatchGetItem` pour tout nouvel élément. Vous ne pouvez pas lire la chronologie à l'aide d'un appel d'API, mais cette solution est plus rentable si les publications peuvent être modifiées fréquemment.

La chronologie est un endroit qui affiche les publications récentes. Nous aurons donc besoin d'un moyen de nettoyer les anciennes. Au lieu d'utiliser une WCU pour les supprimer, vous pouvez utiliser la fonctionnalité Durée de vie de DynamoDB pour le faire gratuitement.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Tous les modèles d'accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
getUserInfoByUserID	Table de base	Requête	PK=<userID>		

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
getFollowerListByUserID	Table de base	Requête	PK=<userID>#follower		
getFollowingListByUserID	Table de base	Requête	PK=<userID>#following		
getPostListByUserID	Table de base	Requête	PK=<userID>#post		
getUserLikesByPostID	Table de base	Requête	PK=<postID>#likelist		
getLikeCountByPostID	Table de base	GetItem	PK=<postID>#likecount		
getTimelineByUserID	Table de base	Requête	PK=<userID>#timeline		

Schéma final du réseau social

Voici la conception du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les exemples DynamoDB sur GitHub.

Table de base :

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour plus d'informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.

3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour plus d'informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour plus d'informations, consultez the section called "Modification d'un modèle existant".
5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Conception de schéma de profil de jeu dans DynamoDB

Cas d'utilisation métier du profil de jeu

Ce cas d'utilisation décrit l'utilisation de DynamoDB pour stocker les profils des joueurs pour un système de jeu. Les utilisateurs (dans ce cas, les joueurs) doivent créer des profils avant de pouvoir interagir avec de nombreux jeux modernes, en particulier les jeux en ligne. Les profils de jeu incluent généralement les éléments suivants :

- Informations de base telles que leur nom d'utilisateur
- Données de jeu telles que les objets et l'équipement
- Enregistrements de jeu tels que les tâches et les activités
- Informations sociales telles que les listes d'amis

Pour satisfaire aux exigences précises en matière d'accès aux requêtes de données de cette application, les clés primaires (clé de partition et clé de tri) utiliseront des noms génériques (PK et SK) afin qu'elles puissent être surchargées de différents types de valeurs, comme nous le verrons ci-dessous.

Les modèles d'accès de cette conception de schéma sont les suivants :

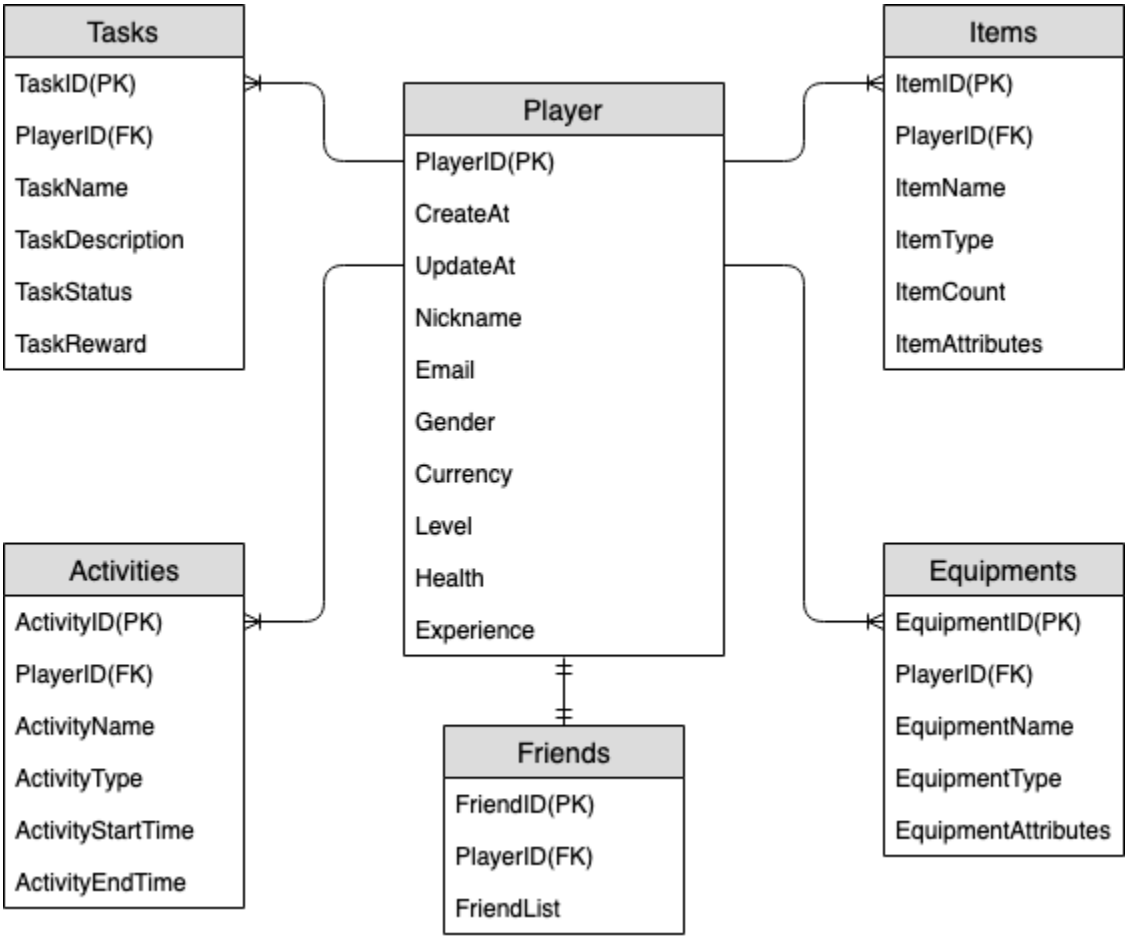
- Obtenir la liste d'amis d'un utilisateur
- Obtenir toutes les informations d'un joueur
- Obtenir la liste d'objets d'un utilisateur
- Obtenir un objet spécifique de la liste d'objets de l'utilisateur
- Mettre à jour le personnage d'un utilisateur

- Mettre à jour le nombre d'objets d'un utilisateur

La taille du profil de jeu varie selon les jeux. La compression des valeurs d'attributs volumineuses permet de maintenir celles-ci dans les limites des éléments dans DynamoDB et de réduire les coûts. La stratégie de gestion du débit dépend de divers facteurs, tels que le nombre de joueurs, le nombre de parties jouées par seconde et la saisonnalité de la charge de travail. Généralement, pour un jeu récent, le nombre de joueurs et le niveau de popularité ne sont pas connus. Nous allons donc commencer par le mode de débit à la demande.

Diagramme des relations entre entités de profil de jeu

Il s'agit du diagramme des relations entre entités (ERD) que nous utiliserons pour la conception du schéma de profil de jeu.



Modèles d'accès de profil de jeu

Voici les modèles d'accès que nous allons prendre en considération pour la conception du schéma de réseau social.

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

Évolution de la conception du schéma de profil de jeu

À partir de l'ERD ci-dessus, nous pouvons voir qu'il s'agit d'un type de relation de type « un à plusieurs » de modélisation de données. Dans DynamoDB, les modèles de données « un à plusieurs » peuvent être organisés en collections d'éléments, ce qui est différent des bases de données relationnelles traditionnelles dans lesquelles plusieurs tables sont créées et liées via des clés étrangères. Une collection d'éléments est un groupe d'éléments qui partagent la même valeur de clé de partition, mais qui ont des valeurs de clé de tri différentes. Au sein d'une collection d'objets, chaque élément possède une valeur de clé de tri unique qui le distingue des autres éléments. Dans cette optique, utilisons le modèle suivant pour les valeurs `HASH` et `RANGE` pour chaque type d'entité.

Pour commencer, nous utilisons des noms génériques tels que `PK` et `SK` pour stocker différents types d'entités dans la même table et ainsi pérenniser le modèle. Pour une meilleure lisibilité, nous pouvons inclure des préfixes pour indiquer le type de données ou inclure un attribut arbitraire nommé `Entity_type` ou `Type`. Dans l'exemple actuel, nous utilisons une chaîne commençant par `player` pour stocker `player_ID` sous `PK`. Nous utilisons ensuite `entity name#` comme préfixe de `SK` et ajoutons un attribut `Type` pour indiquer le type d'entité auquel appartient cette donnée. Cela nous permet de prendre en charge le stockage d'un plus grand nombre de types d'entités à l'avenir et d'utiliser des technologies avancées telles que la surcharge du GSI et le GSI fragmenté pour prendre en charge un plus grand nombre de modèles d'accès.

Commençons à implémenter les modèles d'accès. Les modèles d'accès tels que l'ajout de joueurs et l'ajout d'équipements peuvent être réalisés au cours de l'opération `PutItem`. Nous pouvons donc les ignorer. Dans ce document, nous allons nous concentrer sur les modèles d'accès typiques répertoriés ci-dessus.

Étape 1 : Traitement du modèle d'accès 1 (**`getPlayerFriends`**)

Lors de cette étape, nous traitons le modèle d'accès 1 (`getPlayerFriends`). Dans notre conception actuelle, l'amitié est simple et le nombre d'amis dans le jeu est réduit. Par souci de

simplicité, nous utilisons un type de données de liste pour stocker les listes d'amis (modélisation 1:1). Dans cette conception, nous utilisons `GetItem` pour satisfaire ce modèle d'accès. Au cours de l'opération `GetItem`, nous fournissons explicitement la valeur de la clé de partition et de la clé de tri pour obtenir un élément spécifique.

Toutefois, si un jeu compte un grand nombre d'amis et que les relations entre eux sont complexes (par exemple, les amitiés sont bidirectionnelles avec une composante d'invitation et une composante d'acceptation), il serait nécessaire d'utiliser une relation de type « many-to-many » pour stocker chaque ami individuellement, afin d'atteindre une taille de liste d'amis illimitée. Et si le changement d'amitié implique d'opérer sur plusieurs éléments en même temps, les transactions DynamoDB peuvent être utilisées pour regrouper plusieurs actions et les soumettre sous la forme d'une seule opération radicale `TransactWriteItems` ou `TransactGetItems`.

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
player001	FRIENDS#player001	Type	FriendList
		Friends	[{"M": {"FriendId": {"S":"player002"},"FriendName": {"S":"Alice"}}}, {"M":{"FriendId": {"S":"player003"},"FriendName": {"S":"Bob"}}}]

Étape 2 : Traitement des modèles d'accès 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**) et 4 (**`getPlayerSpecificItem`**)

Lors de cette étape, nous traitons les modèles d'accès 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`) et 4 (`getPlayerSpecificItem`). Ces trois modèles d'accès ont en commun une requête de plage qui utilise l'opération `Query`. En fonction de l'étendue de la requête, une condition de clé et des expressions de filtre sont utilisées. Elles sont couramment utilisées dans le développement pratique.

Dans l'opération Query, nous fournissons une valeur unique pour la clé de partition et nous obtenons tous les éléments avec cette valeur de clé de partition. Le modèle d'accès 2 (`getPlayerAllProfile`) est implémenté de cette manière. Nous pouvons éventuellement ajouter une expression de condition de clé de tri, c'est-à-dire une chaîne qui détermine les éléments à lire à partir de la table. Le modèle d'accès 3 (`getPlayerAllItems`) est implémenté en ajoutant la condition clé de la clé de tri begins_with`ITEMS#`. De plus, afin de simplifier le développement côté application, nous pouvons utiliser des expressions de filtre pour implémenter le modèle d'accès 4 (`getPlayerSpecificItem`).

Voici un exemple de pseudo-code utilisant une expression de filtre qui filtre les éléments de la catégorie `Weapon` :

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
player001	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}

> **Note**
>
> Une expression de filtre est appliquée après la fin de l'opération Query, mais avant que les résultats soient renvoyés au client. Par conséquent, une opération Query utilise la même capacité de lecture, qu'une expression de filtre soit présente ou non.

Si le modèle d'accès consiste à interroger un jeu de données volumineux et à filtrer une grande quantité de données pour ne conserver qu'un petit sous-ensemble de données, l'approche appropriée consiste à concevoir la clé de partition et la clé de tri DynamoDB de manière plus efficace. Par exemple, dans l'exemple ci-dessus pour obtenir un certain `ItemType`, s'il existe de nombreux éléments pour chaque joueur et que la recherche d'un certain `ItemType` est un modèle d'accès type, il serait plus efficace d'importer `ItemType` dans SK sous forme de clé composite. Le modèle de données ressemblerait à ceci : `ITEMS#ItemType#ItemId`.

Étape 3 : Traitement des modèles d'accès 5 (**`updateCharacterAttributes`**) et 6 (**`updateItemCount`**)

Lors de cette étape, nous traitons les modèles d'accès 5 (`updateCharacterAttributes`) et 6 (`updateItemCount`). Lorsque le joueur a besoin de modifier son personnage, pour réduire la

monnaie ou modifier la quantité d'une arme dans ses objets, utilisez `UpdateItem` pour implémenter ces modèles d'accès. Pour mettre à jour la monnaie d'un joueur tout en veillant à ce qu'elle ne descende jamais en dessous d'un montant minimum, nous pouvons ajouter un élément the section called "Exemple de commande CLI" pour réduire le solde uniquement s'il est supérieur ou égal au montant minimum. Voici un exemple de pseudo-code :

```
UpdateExpression: "SET currency = currency - :amount"
ConditionExpression: "currency >= :minAmount"
```

Primary key		Attributes									
Partition key: PK	Sort key: SK	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
player001	#METADATA #player001	Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500 Updated to 500-Amount	10	80	1000

Lors du développement avec DynamoDB et de l'utilisation de compteurs atomiques pour réduire l'inventaire, nous pouvons garantir l'idempotence en utilisant un verrouillage optimiste. Voici un exemple de pseudo-code pour les compteurs atomiques :

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```

Primary key		Attributes				
Partition key: PK	Sort key: SK	Type	ItemName	ItemType	ItemCount	ItemAttributes
player001	ITEMS#001	Item	Health Potion	Consumable	5 Updated to 4	{"M":{"HP": {"N":"50"}}}

De plus, dans un scénario où le joueur achète un objet avec de la monnaie, l'ensemble du processus doit déduire la monnaie et ajouter un objet en même temps. Nous pouvons utiliser les transactions DynamoDB pour regrouper plusieurs actions et les soumettre sous la forme d'une seule opération radicale `TransactWriteItems` ou `TransactGetItems`. `TransactWriteItems` est une opération d'écriture synchrone et idempotente qui regroupe jusqu'à 100 actions d'écriture en une seule opération radicale. Les actions sont exécutées de manière atomique, de sorte qu'elles réussissent toutes ou aucune ne réussit. Les transactions contribuent à éliminer le risque de duplication ou de disparition de monnaie. Pour plus d'informations sur les transactions, consultez Exemple de transactions DynamoDB.

Tous les modèles d'accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
getPlayer Friends	Table de base	GetItem	PK=PlayerID	SK="FRIENDS#playerID"	
getPlayer AllProfile	Table de base	Requête	PK=PlayerID		
getPlayer AllItems	Table de base	Requête	PK=PlayerID	SK begins_with "ITEMS#"	
getPlayer SpecificItem	Table de base	Requête	PK=PlayerID	SK begins_with "ITEMS#"	filterExpression: "ItemType = :itemType " expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	Table de base	UpdateItem	PK=PlayerID	SK="#METADATA#playerID"	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"

Modèle d’accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
updateItemCount	Table de base	UpdateItem	PK=PlayerID	SK =“ITEMS#ItemID”	update-expression: "SET ItemCount = ItemCount - :incr" expression-attribute-values : ’{":incr": {"N":"1"}}’

Schéma final de profil de jeu

Voici la conception du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les exemples DynamoDB sur GitHub.

Table de base :

Primary key		Attributes										
Partition key: PK	Sort key: SK											
player001	#METADATA #player001	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
		Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500	10	80	1000
	ACTIVITY#001	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647475199	Hunting Trip	{"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}}	1647388800	Hunting					
	ACTIVITY#002	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647647999	Mining Adventure	{"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}}	1647561600	Mining					
	ACTIVITY#003	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647820799	Arena Challenge	{"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}}	1647734400	Arena					
	EQUIPMENTS#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}							
	EQUIPMENTS#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}}							
	EQUIPMENTS#003	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Ring of the Mage	Accessory	{"M":{"SP":{"N":"50"}}}							
	FRIENDS#player001	Type	FriendList									
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]									
	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}						
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}						
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}						
	TASK#001	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Find the Lost TreasureFind the Lost Treasure	Get clues from a lost adventurer and find the lost treasure.	InProgress	{"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}}						
	TASK#002	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Defeat Magic Monsters	Go to the Magic Forest and defeat three magic monsters.	Completed	{"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}}						
	TASK#003	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Rescue the Princess	Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King.	Available	{"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}}						

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour de plus amples informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.
3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour de plus amples informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour de plus amples informations, consultez the section called "Modification d'un modèle existant".
5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Conception du schéma d'un système de gestion des réclamations dans DynamoDB

Cas d'utilisation métier d'un système de gestion des réclamations

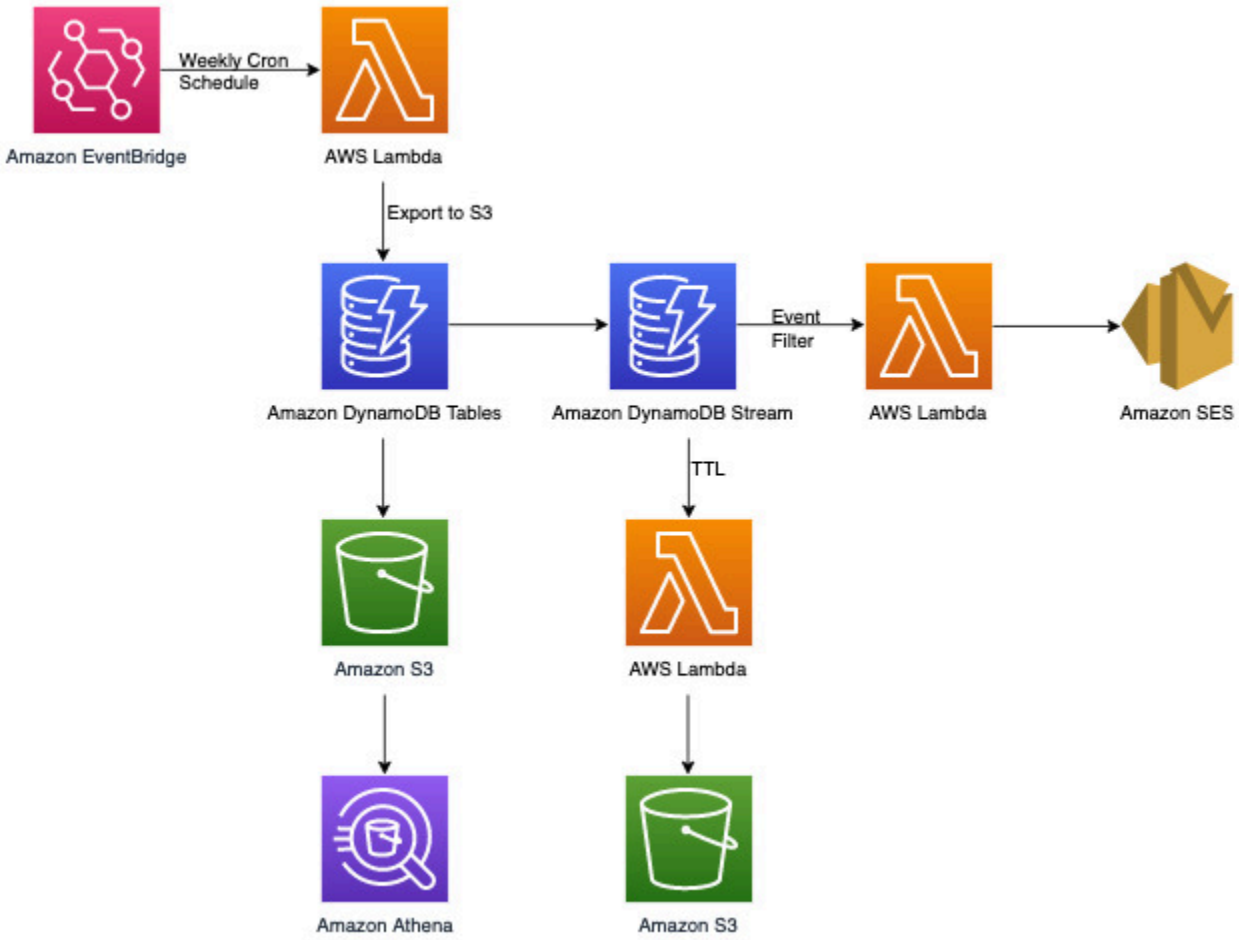
DynamoDB est une base de données parfaitement adaptée à un cas d'utilisation de système de gestion des réclamations (ou centre de contact), car la plupart des modèles d'accès qui y sont associés sont des recherches transactionnelles basées sur des paires clé-valeur. Dans ce scénario, les modèles d'accès types consisteraient à :

- Créer et mettre à jour des réclamations
- Transmettre une réclamation à un échelon supérieur (escalade)
- Créer et lire des commentaires sur une réclamation
- Recueillir toutes les réclamations d'un client
- Recueillir tous les commentaires d'un agent et toutes les escalades

Certains commentaires peuvent s'accompagner de pièces jointes décrivant la réclamation ou la solution. Bien que ces modèles d'accès soient tous de type clé-valeur, d'autres exigences peuvent s'ajouter, comme l'envoi de notifications lorsqu'un nouveau commentaire est ajouté à une réclamation ou l'exécution de requêtes analytiques pour déterminer la répartition hebdomadaire des réclamations par gravité (ou les performances des agents). La nécessité d'archiver les données relatives aux réclamations trois ans après l'enregistrement de la réclamation pourrait constituer une autre exigence liée à la gestion du cycle de vie ou à la conformité.

Diagramme de l'architecture du système de gestion des réclamations

Le schéma suivant illustre l'architecture du système de gestion des réclamations. Ce schéma présente les différentes intégrations d'Service AWS utilisées par le système de gestion des réclamations.



Outre les modèles d'accès transactionnel de type clé-valeur que nous traiterons ultérieurement dans la section sur la modélisation de données DynamoDB, nous sommes en présence de trois exigences

non transactionnelles. Le diagramme d'architecture ci-dessus peut être décomposé en trois flux de travail distincts, à savoir :

1. Envoyer une notification lorsqu'un nouveau commentaire est ajouté à une réclamation
2. Exécuter des requêtes analytiques sur les données hebdomadaires
3. Archiver les données de plus de trois ans

Examinons de plus près chacun d'eux.

Envoyer une notification lorsqu'un nouveau commentaire est ajouté à une réclamation

Nous pouvons utiliser le flux de travail ci-dessous pour répondre à cette exigence :

Flux DynamoDB est un mécanisme de capture des modifications de données qui enregistre toutes les activités d'écriture de vos tables DynamoDB. Vous pouvez configurer des fonctions Lambda pour qu'elles se déclenchent pour tout ou partie de ces modifications. Un filtre d'événements peut être configuré au niveau des déclencheurs Lambda afin de filtrer les événements qui n'ont aucun rapport direct avec le cas d'utilisation. Dans ce cas, nous pouvons utiliser un filtre pour déclencher Lambda uniquement lorsqu'un nouveau commentaire est ajouté et envoyer une notification aux adresses e-mail voulues, qui peuvent être récupérées depuis AWS Secrets Manager ou tout autre magasin d'informations d'identification.

Exécuter des requêtes analytiques sur les données hebdomadaires

DynamoDB est adapté aux charges de travail principalement axées sur le traitement transactionnel en ligne (OLTP). Pour les 10 à 20 % de modèles d'accès restants soumis à des exigences analytiques, les données peuvent être exportées vers S3 à l'aide de la fonctionnalité gérée Exporter vers Amazon S3 sans que cela n'impacte le trafic en direct de la table DynamoDB. Examinez le flux de travail ci-dessous :

Amazon EventBridge peut être utilisé pour déclencher AWS Lambda selon une planification – vous pouvez ainsi configurer une expression cron de sorte que l'invocation Lambda se produise périodiquement. Lambda peut invoquer l'appel de l'API `ExportToS3` et stocker les données DynamoDB dans S3. Un moteur SQL comme Amazon Athena peut ensuite accéder à ces données S3 afin d'exécuter des requêtes analytiques sur les données DynamoDB sans affecter la charge de travail transactionnelle en direct de la table. Voici un exemple de requête Athena qui vise à déterminer le nombre de réclamations par niveau de gravité :

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Cette requête Athena renvoie le résultat suivant :

Results (3)

Search rows

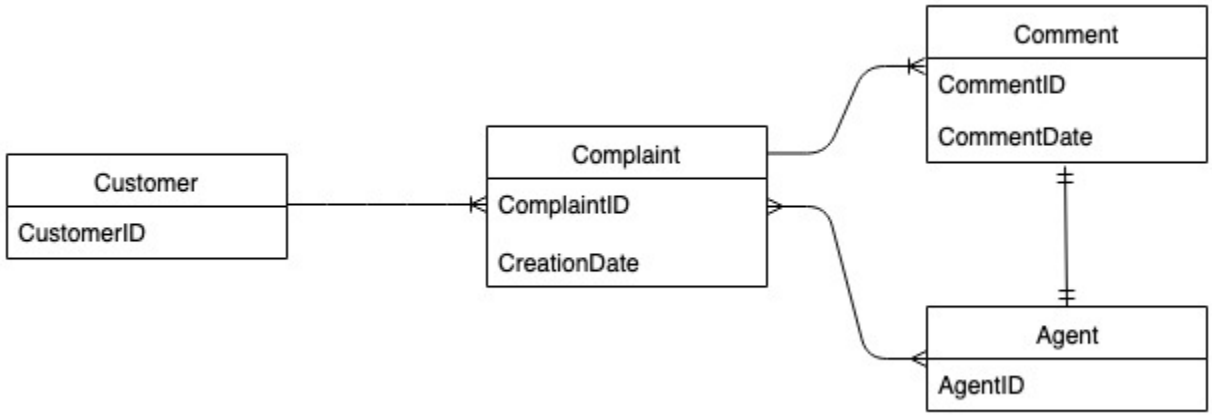
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Archiver les données de plus de trois ans

Vous pouvez tirer parti de la fonctionnalité DynamoDB Durée de vie (TTL) pour supprimer les données obsolètes de votre table DynamoDB sans frais supplémentaires (sauf dans le cas des réplicas de tables globales pour la version (actuelle) 2019.11.21, où les suppressions TTL répliquées dans d'autres régions consomment de la capacité d'écriture). Ces données apparaissent et peuvent être utilisées par des flux DynamoDB pour être archivées dans Amazon S3. Voici comment se présente le flux de travail pour cette exigence :

Diagramme des relations entre entités du système de gestion des réclamations

Il s'agit du diagramme des relations entre entités (ERD) que nous allons utiliser pour la conception du schéma du système de gestion des réclamations.



Modèles d'accès du système de gestion des réclamations

Voici les modèles d'accès que nous allons prendre en considération pour la conception du schéma de gestion des réclamations.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID

4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (de la plus récente à la plus ancienne)
13. getCommentsByAgentID (entre deux dates)

Évolution de la conception du schéma du système de gestion des réclamations

S'agissant d'un système de gestion des réclamations, la plupart des modèles d'accès sont centrés sur la réclamation en tant qu'entité principale. Du fait de sa cardinalité élevée, `ComplaintID` assurera une répartition uniforme des données dans les partitions sous-jacentes, et ce sera également le critère de recherche le plus courant pour nos modèles d'accès identifiés. Par conséquent, il est judicieux d'utiliser `ComplaintID` comme de clé de partition dans cet ensemble de données.

Étape 1 : Traitement des modèles d'accès 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**) et 4 (**`getComplaintByComplaintID`**)

Nous pouvons utiliser une valeur de clé de tri générique appelée « metadata » (ou « AA ») pour stocker les informations propres aux réclamations, telles que `CustomerID`, `State`, `Severity` et `CreationDate`. Nous utilisons des opérations singleton avec `PK=ComplaintID` et `SK="metadata"` pour effectuer les opérations suivantes :

1. `PutItem` pour créer une nouvelle réclamation
2. `UpdateItem` pour mettre à jour la gravité ou d'autres champs dans les métadonnées de la réclamation
3. `GetItem` pour récupérer les métadonnées de la réclamation

Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	assigned	2023-05-10T15:58:00	P2	<Complaint Description>

Étape 2 : Traitement du modèle d'accès 5 (**`addCommentByComplaintID`**)

Ce modèle d'accès nécessite un modèle de relation de type « un-à-plusieurs » entre une réclamation et les commentaires relatifs à la réclamation. Nous allons employer ici la technique du partitionnement vertical pour utiliser une clé de tri et créer une collection d'éléments avec différents types de données. Dans le cas des modèles d'accès 6 (`getAllCommentsByComplaintID`) et 7 (`getLatestCommentByComplaintID`), nous savons que les commentaires devront être triés par ordre chronologique. Nous savons également que plusieurs commentaires pourront être reçus simultanément, ce qui signifie que nous pouvons utiliser la technique de clé de tri composite pour ajouter l'heure et `CommentID` dans l'attribut de clé de tri.

Pour faire face à ce risque de collision de commentaires, il pourrait être envisagé d'accroître la granularité de l'horodatage ou d'ajouter un nombre incrémentiel en guise de suffixe à la place de `Comment_ID`. Dans ce cas, nous allons faire précéder la valeur de clé de tri du préfixe « comm# » pour les éléments correspondant aux commentaires afin de permettre les opérations basées sur une plage.

Nous devons également vérifier que `currentState` dans les métadonnées de la réclamation reflète l'état d'ajout d'un nouveau commentaire. L'ajout d'un commentaire peut indiquer que la réclamation a été affectée à un agent, qu'elle n'a pas été résolue, etc. Afin de regrouper l'ajout de commentaires et la mise à jour de l'état actuel dans les métadonnées de la réclamation, selon une approche du « tout ou rien », nous allons utiliser l'API TransactWriteItems. L'état de la table qui en résulte ressemble désormais à ceci :

Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID
		comm3	2023-05-10T16:00:00	investigating	<Comment text>	AgentB
	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	investigating	2023-05-10T15:58:00	P2	<Complaint Description>

Ajoutons quelques données supplémentaires dans la table et ajoutons également `ComplaintID` sous la forme d'un champ distinct de notre PK pour pérenniser le modèle au cas où nous aurions besoin d'index supplémentaires sur `ComplaintID`. Notez également que

certains commentaires peuvent s'accompagner de pièces jointes que nous stockerons dans Amazon Simple Storage Service et dont nous conserverons uniquement les références ou les URL dans DynamoDB. Dans un souci d'optimisation des coûts et des performances, il est recommandé de garder la base de données transactionnelle aussi légère que possible. Les données ressemblent désormais à ceci :

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Étape 3 : Traitement des modèles d'accès 6 (**`getAllCommentsByComplaintID`**) et 7 (**`getLatestCommentByComplaintID`**)

Pour obtenir tous les commentaires relatifs à une réclamation, nous pouvons utiliser l'opération de requête (`query`) avec la condition `begins_with` au niveau de la clé de tri. Plutôt que de consommer de la capacité de lecture supplémentaire pour lire l'entrée de métadonnées et d'avoir à filtrer les résultats pertinents, cette condition de clé de tri nous permet de lire uniquement ce dont nous avons besoin. Par exemple, une opération de requête avec `PK=Complaint123` et SK begins_with `comm#` renverrait ce qui suit tout en ignorant l'entrée de métadonnées :

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Puisque nous avons besoin du commentaire le plus récent concernant une réclamation du modèle 7 (`getLatestCommentByComplaintID`), utilisons deux paramètres de requête supplémentaires :

1. `ScanIndexForward` doit être défini sur False pour que les résultats soient triés par ordre décroissant
2. `Limit` doit être défini sur 1 pour obtenir le commentaire (unique) le plus récent

Comme pour le modèle d'accès 6 (`getAllCommentsByComplaintID`), nous ignorons l'entrée de métadonnées en utilisant `begins_with comm#` comme condition de clé de tri. À présent, vous pouvez exécuter le modèle d'accès 7 (`PK=Complaint123`) sur cette conception en utilisant l'opération de requête avec `PK=Complaint123` et `SK=begins_with comm#`, `ScanIndexForward=False`, puis `Limit` 1. L'élément ciblé suivant est renvoyé en résultat :

Partition key: PK	Sort key: SK	Attributes					
	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
Complaint123	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Ajoutons d’autres données fictives au tableau.

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text		
		comm4	2022-12-31T19:32:00	waiting	<comm text>		
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>

Étape 4 : Traitement des modèles d'accès 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) et 9 (**`getAllComplaintsByCustomerID`**)

Les modèles d'accès 8 (`getAComplaintbyCustomerIDAndComplaintID`) et 9 (`getAllComplaintsByCustomerID`) introduisent un nouveau critère de recherche : `CustomerID`. Pour le récupérer à partir de la table existante, il faut passer par une opération `Scan` coûteuse afin de lire toutes les données et filtrer ensuite les éléments pertinents pour le `CustomerID` en

question. Nous pouvons améliorer l'efficacité de cette recherche en créant un index secondaire global (GSI) avec `CustomerID` comme clé de partition. Compte tenu de la relation de type « un à plusieurs » qui existe entre le client et les réclamations et du modèle d'accès 9 (`getAllComplaintsByCustomerID`), il serait judicieux d'utiliser `ComplaintID` pour la clé de tri.

Voici comment se présenteraient les données dans le GSI :

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Voici un exemple de requête sur ce GSI pour le modèle d'accès 8 (`getAComplaintbyCustomerIDAndComplaintID`) : `customer_id=custXYZ`, `sort key=Complaint1321`. Le résultat serait le suivant :

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
custXYZ							
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Pour obtenir toutes les réclamations d'un client pour le modèle d'accès 9 (`getAllComplaintsByCustomerID`), la requête sur le GSI serait `customer_id=custXYZ` comme condition de clé de partition. Le résultat serait le suivant :

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Étape 5 : Traitement du modèle d'accès 10 (**`escalateComplaintByComplaintID`**)

Cet accès introduit la notion d'escalade. Pour transmettre une plainte à un échelon supérieur, nous pouvons utiliser `UpdateItem` pour ajouter des attributs tels que `escalated_to` et `escalation_time` à l'élément de métadonnées de réclamation existant. DynamoDB offre une conception de schéma flexible, ce qui signifie qu'un ensemble d'attributs qui ne correspondent pas à une clé peut être uniforme ou discret entre différents éléments. Voici un exemple :

`UpdateItem with PK=Complaint1444, SK=metadata`

Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07

Étape 6 : Traitement des modèles d'accès 11 (**`getAllEscalatedComplaints`**) et 12 (**`getEscalatedComplaintsByAgentID`**)

Sur l'ensemble complet de données, seules quelques réclamations devraient faire l'objet d'une escalade. Par conséquent, la création d'un index sur les attributs liés à l'escalade aboutirait à des recherches efficaces et à un stockage GSI économique. Pour cela, nous pouvons tirer parti de la technique de l'index fragmenté. Voici à quoi ressemblerait le GSI avec la clé de partition `escalated_to` et la clé de tri `escalation_time` :

Primary key		Attributes							
Partition key: escalated_to	**Sort key: escalation_time**								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Pour obtenir toutes les réclamations ayant fait l'objet d'une escalade pour le modèle d'accès 11 (`getAllEscalatedComplaints`), il nous suffit d'analyser ce GSI. Notez que cette analyse sera performante et économique du fait de la taille du GSI. Pour obtenir les réclamations ayant fait l'objet d'une escalade pour un agent déterminé (modèle d'accès 12 (`getEscalatedComplaintsByAgentID`)), la clé de partition serait `escalated_to=agentID` et nous définirions `ScanIndexForward` sur `False` pour un ordre de tri du plus récent au plus ancien.

Étape 7 : Traitement du modèle d'accès 13 (**`getCommentsByAgentID`**)

Pour le dernier modèle d'accès, nous devons effectuer une recherche selon une nouvelle dimension : `AgentID`. Nous avons également besoin d'un ordre chronologique pour lire les commentaires entre deux dates. Nous créons donc un GSI avec `agent_id` comme clé de partition et `comm_date` comme clé de tri. Voici comment se présentent les données dans ce GSI :

Primary key		Attributes					
Partition key: agentID	**Sort key: comm_date**						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Voici un exemple de requête sur ce GSI : `partition key agentID=AgentA` et `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, et en voici le résultat :

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Tous les modèles d’accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d’accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
createComplaint	Table de base	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Table de base	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Table de base	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	Table de base	GetItem	PK=complaint_id	SK=metadata	

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
addCommentByComplaintID	Table de base	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintID	Table de base	Requête	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	Table de base	Requête	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Requête	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Requête	customer_id=customer_id	N/A	
escalateComplaintByComplaintID	Table de base	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Analyser	N/A	N/A	

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
getEscalatedComplaintsByAgentID (de la plus récente à la plus ancienne)	Escalations_GSI	Requête	escalated_to=agent_id	N/A	scan_index_forward=False
getCommentsByAgentID (entre deux dates)	Agents_Comments_GSI	Requête	agent_id=agent_id	SK between (date1, date2)	

Schéma final du système de gestion des réclamations

Voici les conceptions du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les exemples DynamoDB sur GitHub.

Table de base

Primary key		Attributes							
Partition key: PK	**Sort key: SK**								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

Customer_Complaint_GSI

Primary key		Attributes							
Partition key: customer_id	**Sort key: complaint_id**								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

Escalations_GSI

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agents_Comments_GSI

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour de plus amples informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.
3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour de plus amples informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour de plus amples informations, consultez the section called "Modification d'un modèle existant".

5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Conception d'un schéma de paiements récurrents dans DynamoDB

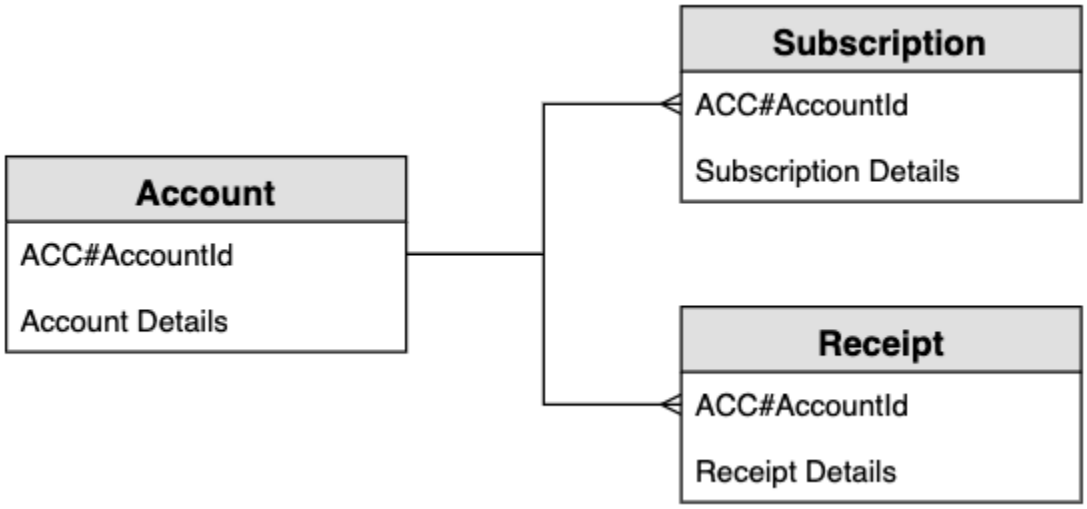
Cas d'utilisation métier de paiements récurrents

Ce cas d'utilisation traite de l'utilisation de DynamoDB pour implémenter un système de paiements récurrents. Le modèle de données comprend les entités suivantes : accounts (comptes), subscriptions (abonnements) et receipts (reçus). Notre cas d'utilisation présente les spécificités suivantes :

- Chaque compte peut être composé de plusieurs abonnements
- L'abonnement présente un `NextPaymentDate` qui correspond à la prochaine date de paiement et un `NextReminderDate`qui correspond à la date où un e-mail de rappel doit être envoyé au client
- Un élément est stocké et mis à jour pour l'abonnement une fois que le paiement a été traité (la taille moyenne des éléments est d'environ 1 Ko et le débit dépend du nombre de comptes et d'abonnements)
- Par ailleurs, le processeur de paiements crée un reçu dans le cadre du processus, qui est stocké dans la table avec un délai d'expiration défini à l'aide d'un attribut TTL

Diagramme des relations entre entités pour les paiements récurrents

Voici le diagramme des relations entre entités (ERD) que nous allons utiliser pour la conception du schéma de paiements récurrents.



Modèles d'accès du système de paiements récurrents

Voici les modèles d'accès que nous allons prendre en considération pour la conception du schéma du système de paiements récurrents.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Conception du schéma pour les paiements récurrents

Les noms génériques `PK` et `SK` sont utilisés pour les attributs de clé afin de permettre le stockage de différents types d'entités dans la même table, comme les entités account, subscription et receipt. Pour commencer, l'utilisateur crée un abonnement et accepte ainsi de payer un montant le même jour de chaque mois en contrepartie d'un produit. Il a la possibilité de choisir le jour du mois auquel le paiement sera effectué. Un rappel sera également envoyé avant l'exécution du paiement. L'application fait appel à deux tâches de traitement par lots qui s'exécutent chaque jour : l'une qui envoie les rappels programmés à cette date et l'autre qui traite les paiements prévus ce même jour.

Étape 1 : Traitement du modèle d'accès 1 (**`createSubscription`**)

Le modèle d'accès 1 (`createSubscription`) est utilisé pour créer dans un premier temps l'abonnement ; les détails comme SKU, `NextPaymentDate`, `NextReminderDate` et `PaymentDetails` sont également définis. Cette étape indique l'état de la table pour un seul compte avec un seul abonnement. Sachant que la collection d'éléments peut inclure plusieurs abonnements, il s'agit d'une relation de type « un-à-plusieurs ».

Primary key		Attributes									
Partition key: PK	Sort key: SK	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	1970-01-01T00:00:00.000Z	2023-05-28	1970-01-01T00:00:00.000Z	2023-05-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Étape 2 : Traitement des modèles d'accès 2 (**`createReceipt`**) et 3 (**`updateSubscription`**)

Le modèle d'accès 2 (`createReceipt`) est utilisé pour créer l'élément de reçu. Une fois que le paiement mensuel a été effectué, le processeur de paiements écrit un reçu dans la table de base. Sachant que la collection d'éléments peut inclure plusieurs reçus, il s'agit d'une relation de type « un-à-plusieurs ». Le processeur de paiements met également à jour l'élément d'abonnement (modèle d'accès 3 (`updateSubscription`)) afin de mettre à jour `NextReminderDate` ou `NextPaymentDate` pour le mois suivant.

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Étape 3 : Traitement du modèle d'accès 4 (**`getDueRemindersByDate`**)

L'application traite les rappels de paiement par lots pour la date du jour. L'application doit donc accéder aux abonnements selon une autre dimension : la date et non le compte. Pour ce cas d'utilisation, il est judicieux d'utiliser un index secondaire global (GSI). Dans cette étape, nous allons ajouter l'index `GSI-1`, qui utilise `NextReminderDate` en guise de clé de partition de GSI. Il n'est pas nécessaire de répliquer tous les éléments. Ce GSI étant un index fragmenté, les éléments de reçus ne sont pas répliqués. Il n'est pas non plus nécessaire de projeter tous les attributs ; il suffit d'inclure un sous-ensemble d'attributs. L'image ci-dessous montre le schéma de `GSI-1`, qui contient les informations dont l'application a besoin pour envoyer l'e-mail de rappel.

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
		SK	PK	SKU	Email	NextPaymentDate
2023-06-21	2023-05-21T14:15:39.247Z	SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Étape 4 : Traitement du modèle d'accès 5 (**`getDuePaymentsByDate`**)

L'application traite les paiements par lots pour la journée en cours de la même manière qu'elle le fait pour les rappels. Nous ajoutons `GSI-2` dans cette étape, et `NextPaymentDate` fait office de clé de partition de GSI. Il n'est pas nécessaire de répliquer tous les éléments. Ce GSI est un index fragmenté, car les éléments de reçus ne sont pas répliqués. L'image ci-dessous montre le schéma de `GSI-2`.

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
		PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
2023-06-28	2023-05-18T14:15:39.247Z	ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Étape 5 : Traitement des modèles d'accès 6 (**`getSubscriptionsByAccount`**) et 7 (**`getReceiptsByAccount`**)

L'application peut récupérer tous les abonnements d'un compte en exécutant une requête sur la table de base qui cible l'identifiant du compte (PK) et utilise l'opérateur de plage pour obtenir tous les éléments pour lesquels SK commence par « SUB# ». De même, l'application peut utiliser la même structure de requête pour récupérer tous les reçus en utilisant un opérateur de plage afin d'obtenir tous les éléments pour lesquels SK commence par « REC# ». Cela nous permet de satisfaire les modèles d'accès 6 (`getSubscriptionsByAccount`) et 7 (`getReceiptsByAccount`). L'application utilise ces modèles d'accès pour permettre aux utilisateurs de consulter leurs abonnements en cours et leurs anciens reçus des six derniers mois. Lors de cette étape, aucune modification n'est apportée au schéma de table, et nous pouvons voir ci-dessous que seuls les éléments d'abonnement sont ciblés dans le modèle d'accès 6 (`getSubscriptionsByAccount`).

Primary key		Attributes										
Partition key: PK	Sort key: SK											
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL						
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200						
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate	
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z	

Tous les modèles d'accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d’accès	Table de base/ GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri
createSubscription	Table de base	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	Table de base	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>
updateSubscription	Table de base	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Requête	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Requête	<NextPaymentDate>	
getSubscriptionsByAccount	Table de base	Requête	ACC#account_id	SK begins_with “SUB#”
getReceiptsByAccount	Table de base	Requête	ACC#account_id	SK begins_with “REC#”

Schéma final pour les paiements récurrents

Voici les conceptions du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les exemples DynamoDB sur GitHub.

Table de base

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

GSI-1

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

GSI-2

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour plus d'informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.
3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour plus d'informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour plus d'informations, consultez the section called "Modification d'un modèle existant".
5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Surveillance des mises à jour du statut d'un appareil dans DynamoDB

Ce cas d'utilisation décrit l'utilisation de DynamoDB pour surveiller les mises à jour du statut d'un appareil (ou les changements d'état d'un appareil) dans DynamoDB.

Cas d'utilisation

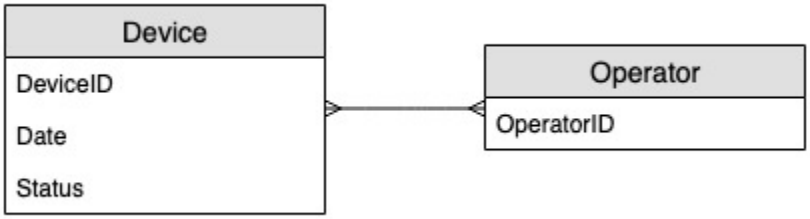
Dans les cas d'utilisation IoT (une fabrique intelligente, par exemple), de nombreux appareils doivent être surveillés par les opérateurs, qui envoient régulièrement leur statut ou leurs journaux à un système de surveillance. En cas de problème avec un appareil, son statut passe de normal à warning. Il existe différents niveaux ou statuts de journalisation en fonction de la gravité et du type de comportement anormal de l'appareil. Le système désigne ensuite un opérateur chargé de vérifier l'appareil et celui-ci peut faire remonter le problème à son superviseur si nécessaire.

Les modèles d'accès types pour ce système incluent :

- Créer une entrée de journal pour un appareil
- Obtenir tous les journaux pour un état d'appareil spécifique en affichant d'abord les journaux les plus récents
- Obtenir tous les journaux pour un opérateur donné entre deux dates
- Obtenir tous les journaux qui ont été remontés pour un superviseur donné
- Obtenir tous les journaux qui ont été remontés avec un état d'appareil spécifique pour un superviseur donné
- Obtenir tous les journaux qui ont été remontés avec un état d'appareil spécifique pour un superviseur donné à une date spécifique

Diagramme des relations entre entités

Voici le diagramme des relations entre entités (ERD) que nous allons utiliser pour surveiller les mises à jour du statut d'un appareil.



Modèles d’accès

Voici les modèles d’accès que nous allons prendre en compte pour surveiller les mises à jour du statut d’un appareil.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Évolution de la conception du schéma

Étape 1 : Traitement des modèles d’accès 1 (**`createLogEntryForSpecificDevice`**) et 2 (**`getLogsForSpecificDevice`**)

L’unité de mise à l’échelle d’un système de suivi des appareils est constituée d’appareils individuels. Dans ce système, un `deviceID` identifie un appareil de manière unique. `deviceID` est donc un bon candidat pour la clé de partition. Chaque appareil envoie régulièrement des informations au système de suivi (toutes les cinq minutes environ). Cet ordre fait de la date un critère de tri logique et, par conséquent, la clé de tri. Dans ce cas, les exemples de données ressembleraient à ce qui suit :

Primary key		Attributes
Partition key: DeviceID	Sort key: State#Date	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL
	2020-04-11T09:25:00	State
		WARNING2
	2020-04-11T09:30:00	State
		NORMAL
d#11223	2020-04-27T16:10:00	State
		WARNING4
	2020-04-27T16:15:00	State
		WARNING4

Pour extraire les entrées de journal d'un appareil spécifique, il est possible d'exécuter une opération query avec la clé de partition `DeviceID="d#12345"`.

Étape 2 : Traitement du modèle d'accès 3 (**`getWarningLogsForSpecificDevice`**)

Étant donné que `State` est un attribut non clé, le traitement du modèle d'accès 3 avec le schéma actuel nécessiterait une expression de filtre. Dans DynamoDB, les expressions de filtre sont appliquées après la lecture des données à l'aide d'expressions de condition de clé. Par exemple, si nous devions extraire les journaux d'avertissement pour `d#12345`, l'opération de requête avec la clé de partition `DeviceID="d#12345"` lirait quatre éléments du tableau ci-dessus, puis filtrerait le seul élément sans l'état warning. Cette approche n'est pas efficace à grande échelle. L'expression de filtre peut être un bon moyen d'exclure les éléments interrogés si le ratio des éléments exclus est

faible ou si la requête est rarement exécutée. Toutefois, dans les cas où de nombreux éléments sont extraits d'une table et que la majorité d'entre eux sont filtrés, nous pouvons continuer à faire évoluer la conception de notre table afin qu'elle s'exécute plus efficacement.

Nous allons changer la façon de traiter ce modèle d'accès en utilisant des clés de tri composites. Vous pouvez importer des exemples de données à partir de DeviceStateLog_3.json où la clé de tri est remplacée par `State#Date`. Cette clé de tri est la composition des attributs `State`, # et `Date`. Dans cet exemple, # est utilisé comme délimiteur. Les données ressemblent désormais à ce qui suit :

Primary key	
Partition key: DeviceID	**Sort key: State#Date**
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00

Pour extraire uniquement les journaux d'avertissement d'un appareil, la requête devient plus ciblée avec ce schéma. La condition de clé pour la requête utilise la clé de partition `DeviceID="d#12345"` et la clé de tri `State#Date begins_with "WARNING"`. Cette requête ne lira que les trois éléments pertinents avec l'état warning.

Étape 3 : Traitement du modèle d'accès 4 (**`getLogsForOperatorBetweenTwoDates`**)

Vous pouvez importer DeviceStateLog_4.json où l'attribut `Operator` a été ajouté à la table `DeviceStateLog` avec des exemples de données.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State
		Liz	2020-04-24T14:55:00	NORMAL
	WARNING1#2020-04-24T14:40:00	Operator	Date	State
		Liz	2020-04-24T14:40:00	WARNING1
	WARNING1#2020-04-24T14:45:00	Operator	Date	State
		Liz	2020-04-24T14:45:00	WARNING1
	WARNING1#2020-04-24T14:50:00	Operator	Date	State
		Liz	2020-04-24T14:50:00	WARNING1
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State
		Liz	2020-04-11T06:00:00	NORMAL
	NORMAL#2020-04-11T09:30:00	Operator	Date	State
		Sue	2020-04-11T09:30:00	NORMAL
	WARNING2#2020-04-11T09:25:00	Operator	Date	State
		Sue	2020-04-11T09:25:00	WARNING2
	WARNING3#2020-04-11T05:50:00	Operator	Date	State
		Sue	2020-04-11T05:50:00	WARNING3
	WARNING3#2020-04-11T05:55:00	Operator	Date	State
		Liz	2020-04-11T05:55:00	WARNING3
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State
		Sue	2020-04-27T16:10:00	WARNING4
	WARNING4#2020-04-27T16:15:00	Operator	Date	State
		Sue	2020-04-27T16:15:00	WARNING4

Étant donné que `Operator` n'est pas une clé de partition actuellement, il n'existe aucun moyen d'effectuer une recherche directe clé-valeur dans cette table en fonction de `OperatorID`. Nous

devrons créer une collection d'éléments avec un index secondaire global sur `OperatorID`. Étant donné que le modèle d'accès nécessite une recherche basée sur les dates, Date est donc l'attribut de clé de tri pour l'index secondaire global (GSI). Voici à quoi ressemble désormais le GSI :

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Pour le modèle d'accès 4 (`getLogsForOperatorBetweenTwoDates`), vous pouvez interroger ce GSI avec la clé de partition `OperatorID=Liz` et la clé de tri `Date` entre `2020-04-11T05:58:00` et `2020-04-24T14:50:00`.

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
Liz	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Étape 4 : Traitement des modèles d'accès 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**) et 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**)

Nous utiliserons un index fragmenté pour traiter ces modèles d'accès.

Les index secondaires globaux sont fragmentés par défaut, de sorte que seuls les éléments de la table de base contenant les attributs de clé primaire de l'index apparaîtront réellement dans l'index. Il s'agit d'une autre façon d'exclure les éléments qui ne sont pas pertinents pour le modèle d'accès modélisé.

Vous pouvez importer DeviceStateLog_6.json où l'attribut `EscalatedTo` a été ajouté à la table `DeviceStateLog` avec des exemples de données. Comme mentionné précédemment, tous les journaux ne font pas l'objet d'une remontée à un superviseur.

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

Vous pouvez maintenant créer un GSI où `EscalatedTo` est la clé de partition et `State#Date` est la clé de tri. Notez que seuls les éléments qui possèdent à la fois les attributs `EscalatedTo` et `State#Date` apparaissent dans l'index.

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date	DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

Les autres modèles d'accès sont résumés comme suit :

Tous les modèles d'accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
createLogEntryForSpecificDevice	Table de base	PutItem	DeviceID= deviceId	State#Date=state#date	
getLogsForSpecificDevice	Table de base	Requête	DeviceID= deviceId	State#Date begins_with "state1#"	ScanIndexForward = False
getWarningLogsForSpecificDevice	Table de base	Requête	DeviceID= deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Requête	Operator= operatorName	Date between date1 and date2	
getEscalatedLogsForSupervisor	GSI-2	Requête	EscalatedTo=supervisorName		

Modèle d'accès	Table de base/GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri	Autres conditions/ filtres
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Requête	EscalatedTo=supervisorName	State#Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Requête	EscalatedTo=supervisorName	State#Date begins_with "state1#date1"	

Schéma final

Voici les conceptions du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les exemples DynamoDB sur GitHub.

Table de base

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date	DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour de plus amples informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.
3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour de plus amples informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour de plus amples informations, consultez the section called "Modification d'un modèle existant".
5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Utilisation de DynamoDB comme magasin de données pour un magasin en ligne

Ce cas d'utilisation décrit l'utilisation de DynamoD comme magasin de données pour une boutique en ligne (ou boutique virtuelle).

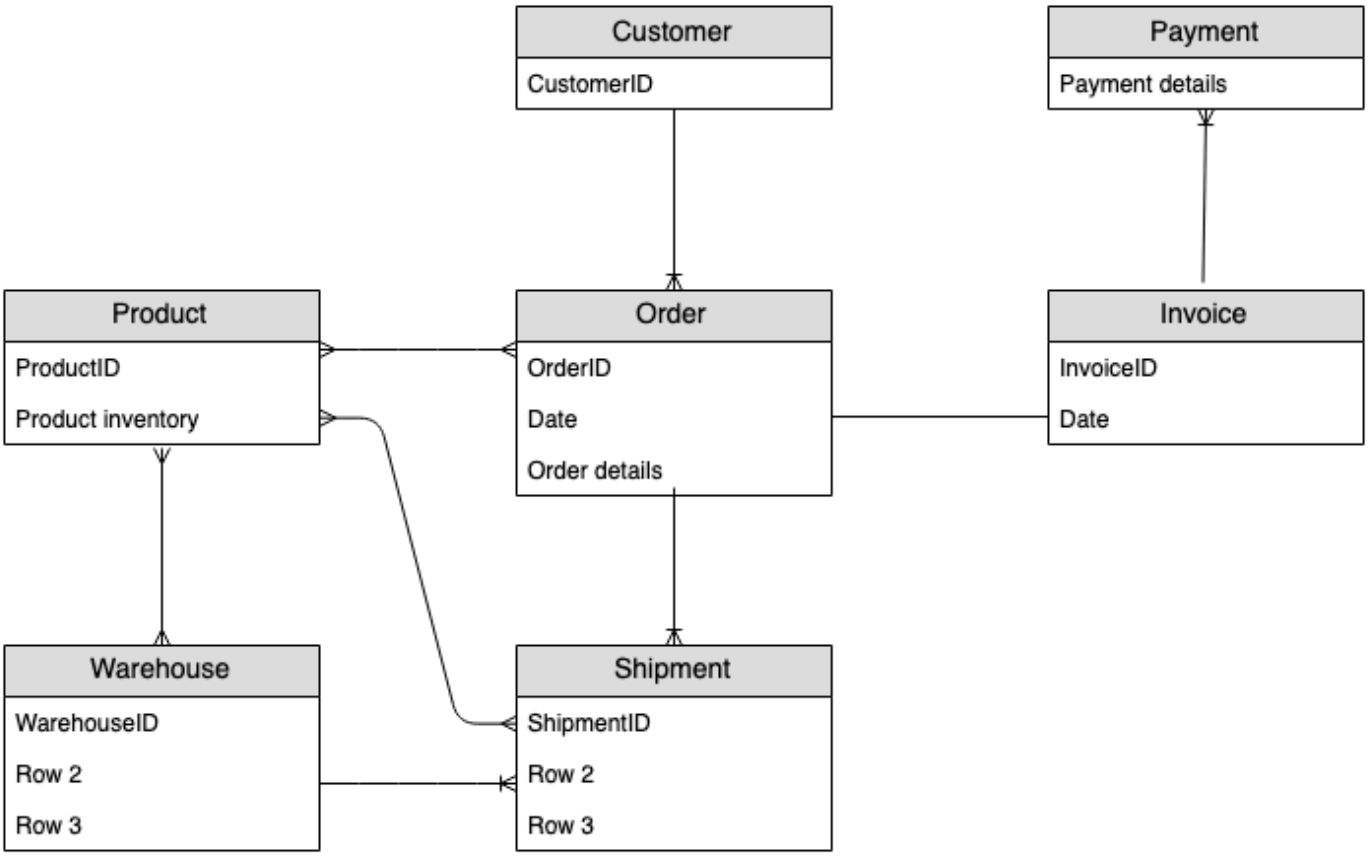
Cas d’utilisation

Une boutique en ligne permet aux utilisateurs de parcourir différents produits et de les acheter par la suite. Sur la base de la facture générée, le client peut payer à l’aide d’un code de réduction ou d’une carte-cadeau, puis régler le montant restant par carte de crédit. Les produits achetés seront collectés dans l’un des nombreux entrepôts et seront expédiés à l’adresse indiquée. Les modèles d’accès types d’une boutique en ligne incluent :

- Obtenir un client pour un customerId donné
- Obtenir un produit pour un productId donné
- Obtenir un entrepôt pour un warehouseId donné
- Obtenez un inventaire de produits pour tous les entrepôts à l’aide d’un productId
- Obtenir une commande pour un orderId donné
- Obtenir tous les produits pour un orderId donné
- Obtenir une facture pour un orderId donné
- Obtenir toutes les expéditions pour un orderId donné
- Obtenir toutes les commandes pour un productId donné et une plage de dates donnée
- Obtenir une facture pour un invoiceId donné
- Obtenir tous les paiements pour un invoiceId donné
- Obtenir les détails d’une expédition pour un shipmentId donné
- Obtenir toutes les expéditions pour un warehouseId donné
- Obtenir l’inventaire de tous les produits pour un warehouseId donné
- Obtenir toutes les factures pour un customerId donné et une plage de dates donnée
- Obtenir tous les produits commandés par un customerId donné pour une plage de dates donnée

Diagramme des relations entre entités

Voici le diagramme des relations entre entités (ERD) que nous utiliserons pour modéliser DynamoDB en tant que magasin de données pour une boutique en ligne.



Modèles d’accès

Voici les modèles d’accès que nous allons prendre en compte lors de l’utilisation de DynamoDB comme magasin de données pour une boutique en ligne.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`

10getInvoiceByInvoiceId

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

Évolution de la conception du schéma

À l'aide de NoSQL Workbench pour DynamoDB, importez AnOnlineShop_1.json pour créer un modèle de données appelé `AnOnlineShop` et une table appelée `OnlineShop`. Notez que nous utilisons les noms génériques `PK` et `SK` pour la clé de partition et la clé de tri. Cette méthode permet de stocker différents types d'entités dans la même table.

Étape 1 : Traitement du modèle d'accès 1 (**`getCustomerByCustomerId`**)

Importez AnOnlineShop_2.json pour traiter le modèle d'accès 1 (`getCustomerByCustomerId`). Étant donné que certaines entités n'ont aucune relation avec d'autres entités, nous utiliserons la même valeur de `PK` et `SK` pour ces entités. Dans les exemples de données, notez que les clés utilisent le préfixe `c#` afin de distinguer `customerId` des autres entités qui seront ajoutées ultérieurement. Cette méthode est également répétée pour d'autres entités.

Pour traiter ce modèle d'accès, il est possible d'utiliser l'opération `GetItem` avec `PK=customerId` et `SK=customerId`.

Étape 2 : Traitement du modèle d'accès 2 (**`getProductByProductId`**)

Importez AnOnlineShop_3.json pour traiter le modèle d'accès 2 (`getProductByProductId`) pour l'entité `product`. Les entités de produit utilisent le préfixe `p#` et le même attribut de clé de tri a été utilisé pour stocker `customerID` et `productID`. Un nom générique et le partitionnement vertical nous permettent de créer ces ensembles d'éléments pour une conception de table unique efficace.

Pour traiter ce modèle d'accès, il est possible d'utiliser l'opération `GetItem` avec `PK=productId` et `SK=productId`.

Étape 3 : Traitement du modèle d'accès 3 (**`getWarehouseByWarehouseId`**)

Importez AnOnlineShop_4.json pour traiter le modèle d'accès 3 (`getWarehouseByWarehouseId`) pour l'entité `warehouse`. Les entités `customer`, `product` et `warehouse` ont actuellement été ajoutées à la même table. Ils se distinguent par l'utilisation de préfixes et de l'attribut `EntityType`. Un attribut de type (ou l'utilisation d'un préfixe) améliore la lisibilité du modèle. Il serait moins lisible si nous stockions simplement les ID alphanumériques de différentes entités dans le même attribut. Il serait difficile de distinguer une entité d'une autre en l'absence de ces identifiants.

Pour traiter ce modèle d'accès, il est possible d'utiliser l'opération `GetItem` avec `PK=warehouseId` et `SK=warehouseId`.

Table de base :

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Étape 4 : Traitement du modèle d'accès 4 (**`getProductInventoryByProductId`**)

Importez AnOnlineShop_5.json pour traiter le modèle d'accès 4 (`getProductInventoryByProductId`). L'entité `warehouseItem` est utilisée pour suivre le nombre de produits dans chaque entrepôt. Cet élément est habituellement mis à jour lorsqu'un produit est ajouté ou retiré d'un entrepôt. Comme on peut le voir dans l'ERD, il existe une relation de type "many-to-many" entre `product` et `warehouse`. Ici, la relation de type "un à plusieurs" entre `product` et `warehouse` est modélisée comme `warehouseItem`. Plus tard, la relation de type "un à plusieurs" entre `warehouse` et `product` sera également modélisée.

Le modèle d'accès 4 peut être traité à l'aide d'une requête sur `PK=ProductId` et `SK begins_with "w#"`.

Pour plus d'informations sur `begins_with()` et d'autres expressions pouvant être appliquées aux clés de tri, consultez Expressions de condition de clé.

Table de base :

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
p#12345	w#12345	EntityType	Quantity	
		warehouseItem	50	
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Étape 5 : Traitement des modèles d'accès 5 (**`getOrderDetailsByOrderId`**) et 6 (**`getProductByOrderId`**)

Ajoutez d'autres éléments `customer`, `product` et `warehouse` à la table en important AnOnlineShop_6.json. Ensuite, importez AnOnlineShop_7.json pour créer un ensemble d'éléments pour `order` qui peut traiter les modèles d'accès 5 (`getOrderDetailsByOrderId`) et 6 (`getProductByOrderId`). La relation de type "un à plusieurs" entre `order` et `product` est modélisée sous forme d'entités orderItem.

Pour traiter le modèle d'accès 5 (`getOrderDetailsByOrderId`), interrogez la table avec `PK=orderId`. Cette requête fournira toutes les informations relatives à la commande, y compris le `customerId` et les produits commandés.

Table de base :

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Pour traiter le modèle d'accès 6 (`getProductByOrderId`), nous devons lire les produits uniquement dans `order`. Pour ce faire, interrogez la table avec `PK=orderId` et `SK begins_with "p#"`.

Table de base :

Primary key		Attributes		
Partition key: PK	Sort key: SK			
	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
o#12345	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Étape 6 : Traitement du modèle d'accès 7 (**`getInvoiceByOrderId`**)

Importez AnOnlineShop_8.json pour ajouter une entité `invoice` à l'ensemble d'éléments order pour traiter le modèle d'accès 7 (`getInvoiceByOrderId`). Pour traiter ce modèle d'accès, il est possible d'utiliser une opération query avec `PK=orderId` et `SK begins_with "i#"`.

Table de base :

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	i#55443	EntityType	Amount	Date
		invoice	400	2020-06-21T19:18:00
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Étape 7 : Traitement du modèle d'accès 8 (**`getShipmentByOrderId`**)

Importez AnOnlineShop_9.json pour ajouter les entités `shipment` à l'ensemble d'éléments order pour traiter le modèle d'accès 8 (`getShipmentByOrderId`). Nous développons le même modèle partitionné verticalement en ajoutant d'autres types d'entités dans la conception de table unique. Vous pouvez remarquer que l'ensemble d'éléments order contient les différentes relations d'une entité `order` avec les entités `shipment`, `orderItem` et `invoice`.

Pour obtenir les expéditions par `orderId`, vous pouvez effectuer une opération query avec `PK=orderId` et `SK begins_with "sh#"`.

Table de base :

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount	Date		
		invoice	400	2020-06-21T19:18:00		
	p#12345	EntityType	Price	Quantity		
		orderItem	100	2		
	p#99887	EntityType	Price	Quantity		
		orderItem	40	5		
	sh#88899	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00	w#12376
	sh#98765	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00	w#12345

Étape 8 : Traitement du modèle d'accès 9 (**`getOrderByProductIdForDateRange`**)

Nous avons créé un ensemble d'éléments order à l'étape précédente. Ce modèle d'accès comporte de nouvelles dimensions de recherche (`ProductID` et `Date`) qui vous obligent à analyser l'ensemble de la table et à filtrer les enregistrements pertinents pour extraire les éléments ciblés. Pour traiter ce modèle d'accès, nous allons devoir créer un index secondaire global (GSI). Importez AnOnlineShop_10.json pour créer un ensemble d'éléments à l'aide du GSI qui permet de récupérer les données `orderItem` issues de plusieurs ensembles d'éléments order. Les données comportent désormais `GSI1-PK` et `GSI1-SK`, qui seront la clé de partition et la clé de tri de `GSI1`, respectivement.

DynamoDB renseigne automatiquement les éléments qui contiennent les attributs de clé d'un GSI, de la table au GSI. Il n'est pas nécessaire d'effectuer des insertions manuelles supplémentaires dans le GSI.

Pour traiter le modèle d'accès 9, exécutez une requête sur `GSI1` avec `GSI1-PK=productId` et `GSI1SK between (date1, date2)`.

Table de base :

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2020-06-21T19:18:00	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2020-06-21T19:20:00	40	5

GSI1 :

Primary key		Attributes				
Partition key: GSI1-PK	**Sort key: GSI1-SK**					
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#12345	orderItem	2	100
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#99887	orderItem	5	40

Étape 9 : Traitement des modèles d'accès 10 (**`getInvoiceByInvoiceId`**) et 11 (**`getPaymentByInvoiceId`**)

Importez AnOnlineShop_11.json pour traiter les modèles d'accès 10 (`getInvoiceByInvoiceId`) et 11 (`getPaymentByInvoiceId`), qui sont tous les deux liés à `invoice`. Même s'il s'agit de deux modèles d'accès différents, ils sont réalisés en utilisant la même condition de clé. `Payments` est défini sous la forme d'un attribut avec le type de données de carte sur l'entité `invoice`.

> **Note**
>
> `GSI1-PK` et `GSI1-SK` sont surchargés pour stocker des informations sur différentes entités afin que plusieurs modèles d'accès puissent être utilisés à partir du même GSI. Pour plus

d'informations sur la surcharge du GSI, consultez Surcharge des index secondaires globaux dans DynamoDB.

Pour traiter le modèle d'accès 10 et 11, interrogez `GSI1` avec `GSI1-PK=invoiceId` et `GSI1-SK=invoiceId`.

GSI1 :

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
i#55443	i#55443	o#12345	i#55443	invoice	c#12345	i#2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00

Étape 10 : Traitement des modèles d'accès 12 (**`getShipmentDetailsByShipmentId`**) et 13 (**`getShipmentByWarehouseId`**)

Importez AnOnlineShop_12.json pour traiter les modèles d'accès 12 (`getShipmentDetailsByShipmentId`) et 13 (`getShipmentByWarehouseId`).

Vous pouvez remarquer que les entités `shipmentItem` sont ajoutées à l'ensemble d'éléments order sur la table de base afin de pouvoir récupérer tous les détails d'une commande en une seule opération query.

Table de base :

Primary key		Attributes							
Partition key: PK	Sort key: SK								
o#12345	sh#88899	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#88899	sh#88899	w#12376	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
	sh#98765	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#98765	sh#98765	w#12345	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00
	shp#12345	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#99887	3				
	shp#54321	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#88899	p#99887	2				
	shp#55555	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#12345	2				

Les clés de partition et de tri `GSI1` ont déjà été utilisées pour modéliser une relation de type "un à plusieurs" entre `shipment` et `shipmentItem`. Pour traiter le modèle d'accès 12 (`getShipmentDetailsByShipmentId`), interrogez `GSI1` avec `GSI1-PK=shipmentId` et `GSI1-SK=shipmentId`.

GSI1 :

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
sh#88899	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
sh#98765	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

Nous devrons créer un autre GSI (`GSI2`) pour modéliser la relation de type "un à plusieurs" entre `warehouse` et `shipment` pour le modèle d'accès 13 (`getShipmentByWarehouseId`). Pour traiter

ce modèle d'accès, interrogez `GSI2` avec `GSI2-PK=warehouseId` et `GSI2-SK begins_with "sh#"`.

GSI2 :

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12376	sh#88899	o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12345	sh#98765	o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00

Étape 11 : Traitement des modèles d'accès 14 (**`getProductInventoryByWarehouseId`**), 15 (**`getInvoiceByCustomerIdForDateRange`**) et 16 (**`getProductsByCustomerIdForDateRange`**)

Importez AnOnlineShop_13.json pour ajouter des données relatives au prochain ensemble de modèles d'accès. Pour traiter le modèle d'accès 14 (`getProductInventoryByWarehouseId`), interrogez `GSI2` avec `GSI2-PK=warehouseId` et `GSI2-SK begins_with "p#"`.

GSI2 :

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Pour traiter le modèle d'accès 15 (`getInvoiceByCustomerIdForDateRange`), interrogez `GSI2` avec `GSI2-PK=customerId` et `GSI2-SK between (i#date1, i#date2)`.

GSI2 :

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Pour traiter le modèle d'accès 16 (`getProductsByCustomerIdForDateRange`), interrogez `GSI2` avec `GSI2-PK=customerId` et `GSI2-SK between (p#date1, p#date2)`.

GSI2 :

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Note

Dans NoSQL Workbench, les facettes représentent les différents modèles d'accès aux données d'une application pour DynamoDB. Les facettes vous permettent d'afficher un sous-ensemble de données dans une table, sans avoir à voir les enregistrements qui ne répondent pas aux contraintes de la facette. Les facettes sont considérées comme un outil de modélisation de données visuelles et n'existent pas en tant que construction utilisable dans DynamoDB, car elles sont purement une aide à la modélisation des modèles d'accès. Importez AnOnlineShop_facets.json pour voir les facettes de ce cas d'utilisation.

Tous les modèles d'accès et la façon dont ils sont traités par la conception du schéma sont résumés dans le tableau ci-dessous :

Modèle d'accès	Table de base/ GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri
getCustomerByCustomerId	Table de base	GetItem	PK=customerId	SK=customerId
getProductByProductId	Table de base	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Table de base	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Table de base	Requête	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Table de base	Requête	PK=orderId	
getProductByOrderId	Table de base	Requête	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Table de base	Requête	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Table de base	Requête	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Requête	PK=productId	SK between date1 and date2
getInvoiceByInvoiceId	GSI1	Requête	PK=invoiceId	SK=invoiceId

Modèle d'accès	Table de base/ GSI/LSI	Opération	Valeur de la clé de partition	Valeur de clé de tri
getPaymentByInvoiceId	GSI1	Requête	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Requête	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Requête	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Requête	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Requête	PK=customerId	SK between i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Requête	PK=customerId	SK between p#date1 and p#date2

Schéma final pour la boutique en ligne

Voici les conceptions du schéma final. Pour télécharger cette conception de schéma au format JSON, consultez les modèles de conception DynamoDB sur GitHub.

Table de base

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
		warehouseItem	4		
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}		
	w#99887	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":		

GSI1

Primary key		Attributes							
Partition key: GSI1-PK	**Sort key: GSI1-SK**								
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#12345	orderItem	c#12345	2020-06-21T19:18:00	100	2	
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#99887	orderItem	c#12345	2020-06-21T19:20:00	40	5	
i#55443	i#55443	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	c#12345	2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

GSI2

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
	sh#98765	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
c#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	
w#12376	sh#88899	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00

Utilisation de NoSQL Workbench avec cette conception de schéma

Vous pouvez importer ce schéma final dans NoSQL Workbench, un outil visuel qui fournit des fonctionnalités de modélisation des données, de visualisation des données et de développement des requêtes pour DynamoDB, afin d'explorer et de modifier davantage votre nouveau projet. Pour commencer, procédez comme suit :

1. Téléchargez NoSQL Workbench. Pour plus d'informations, consultez the section called "Téléchargement".
2. Téléchargez le fichier de schéma JSON répertorié ci-dessus, qui est déjà au format du modèle NoSQL Workbench.
3. Importez le fichier de schéma JSON dans NoSQL Workbench. Pour plus d'informations, consultez the section called "Importation d'un modèle existant".
4. Une fois que vous l'avez importé dans NoSQL Workbench, vous pouvez modifier le modèle de données. Pour plus d'informations, consultez the section called "Modification d'un modèle existant".
5. Pour visualiser votre modèle de données, ajouter des exemples de données ou importer des exemples de données à partir d'un fichier CSV, utilisez la fonctionnalité Visualiseur de données de NoSQL Workbench.

Bonnes pratiques de modélisation des données relationnelles dans DynamoDB

Cette section présente les bonnes pratiques de modélisation des données relationnelles dans Amazon DynamoDB Tout d'abord, nous présentons les concepts traditionnels de modélisation des données. Ensuite, nous décrivons les avantages liés à l'utilisation de DynamoDB par rapport aux systèmes de gestion de base de données relationnelle traditionnels, en expliquant en quoi cela dispense d'utiliser des opérations JOIN et réduit les frais généraux.

Nous expliquons ensuite comment concevoir une table DynamoDB qui se met à l'échelle de manière efficace. Enfin, nous proposons un exemple de modélisation de données relationnelles dans DynamoDB.

Rubriques

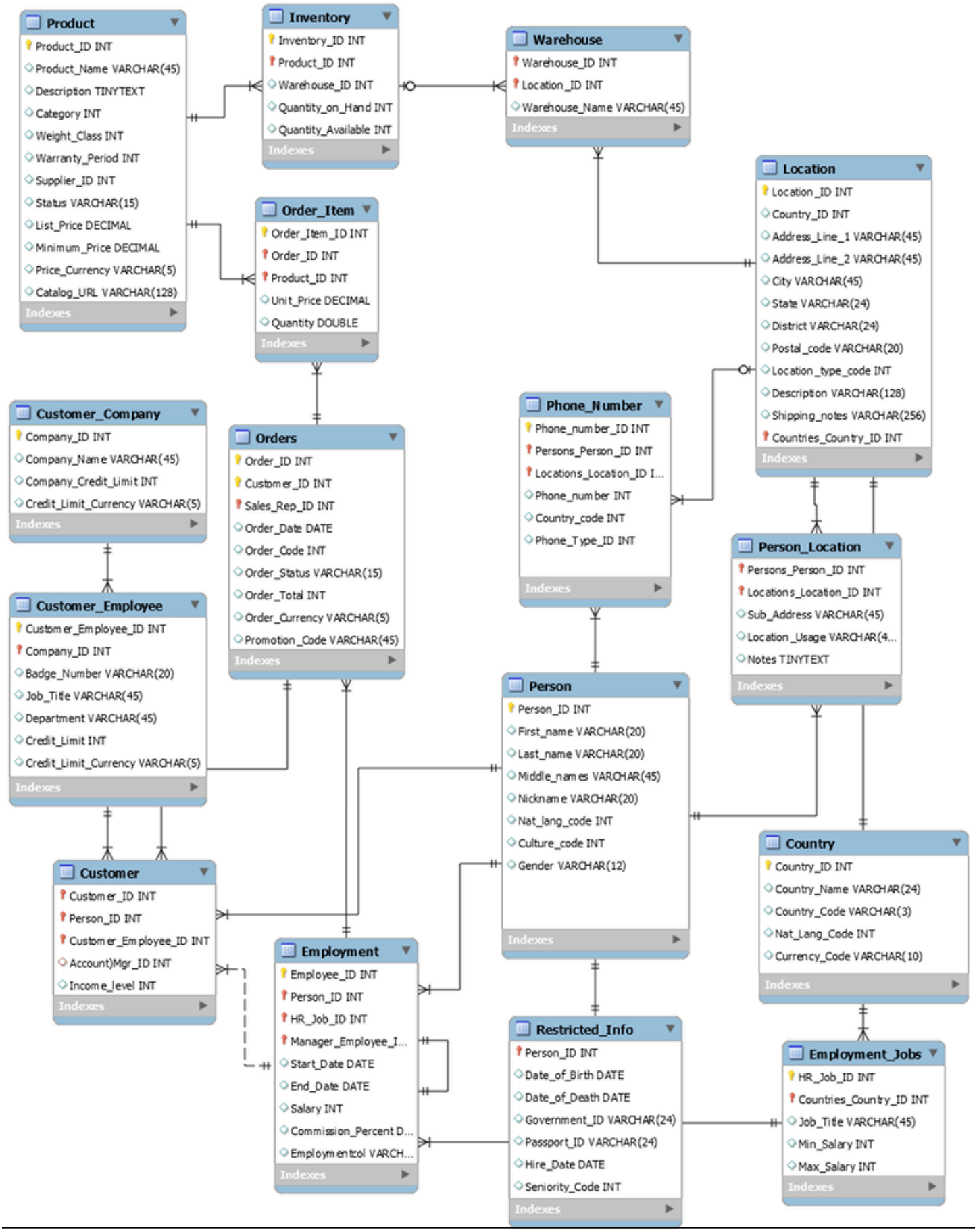
- Modèles de base de données relationnelle traditionnels
- Comment DynamoDB élimine le besoin d'opérations JOIN

- Comment les transactions DynamoDB éliminent la surcharge pour le processus d'écriture
- Premiers pas pour la modélisation des données relationnelles dans DynamoDB
- Exemple de modélisation des données relationnelles dans DynamoDB

Modèles de base de données relationnelle traditionnels

Un système de gestion de base de données relationnelle (SGBDR) traditionnel stocke les données dans une structure relationnelle normalisée. L'objectif du modèle de données relationnel est de réduire la duplication des données (par le biais de la normalisation) afin de garantir l'intégrité référentielle et de réduire les anomalies des données.

Le schéma suivant est un exemple de modèle de données relationnel pour une application de saisie de commandes générique. Cette application prend en charge un schéma de ressources humaines qui soutient les systèmes de support d'exploitation et d'activités d'un fabricant fictif.


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

En tant que service de base de données non relationnelle, DynamoDB offre de nombreux avantages par rapport aux systèmes de gestion de base de données relationnelle traditionnels.

Comment DynamoDB élimine le besoin d'opérations JOIN

Un SGBDR utilise un langage SQL (Structure Query Language) pour renvoyer les données à l'application. Du fait de la normalisation du modèle de données, les requêtes de ce type nécessitent généralement l'utilisation de l'opérateur `JOIN` pour associer les données d'une ou plusieurs tables.

Par exemple, pour générer une liste d'articles de bon de commande triés selon la quantité en stock dans tous les entrepôts pouvant livrer chacun d'entre eux, vous pouvez exécuter la requête SQL suivante sur le schéma précédent.

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

Les requêtes SQL de ce type peuvent fournir une API flexible pour accéder aux données, mais elles nécessitent un volume de traitement important. Chaque jointure contenue dans la requête accentue la complexité d'exécution de la requête, car les données de chaque table doivent être préparées, puis assemblées pour renvoyer le jeu de résultats.

La taille des tables et la présence d'index dans les colonnes jointes sont d'autres facteurs qui peuvent avoir une incidence sur le temps d'exécution des requêtes. La requête précédente initie des requêtes complexes sur plusieurs tables, puis trie le jeu de résultats.

L'élimination du besoin de `JOIN`s est au cœur de la modélisation des données NoSQL. C'est pourquoi nous avons conçu DynamoDB pour qu'il soit compatible avec Amazon.com, et c'est pourquoi DynamoDB peut fournir des performances constantes à n'importe quelle échelle. Compte tenu de la complexité d'exécution des requêtes SQL et des `JOIN`s performances du SGBDR ne sont pas constantes à grande échelle. Cela entraîne des problèmes de performances à mesure que les applications des clients se développent.

Bien que la normalisation des données réduise la quantité de données stockées sur le disque, les ressources les plus limitées qui ont un impact sur les performances sont souvent le temps processeur et la latence du réseau.

DynamoDB est conçu pour minimiser ces deux contraintes en éliminant `JOINs` (et en encourageant la dénormalisation des données) et en optimisant l'architecture de la base de données afin de répondre entièrement à une requête d'application avec une seule requête pour un élément. Ces qualités permettent à DynamoDB d'offrir des performances en millisecondes à un chiffre à n'importe quelle échelle. Cela s'explique par le fait que la complexité d'exécution des opérations DynamoDB est constante, quelle que soit la taille des données, pour les modèles d'accès courants.

Comment les transactions DynamoDB éliminent la surcharge pour le processus d'écriture

L'utilisation de transactions pour écrire dans un schéma normalisé est un autre facteur de ralentissement d'un SGBDR. Comme indiqué dans l'exemple, les structures de données relationnelles utilisées par la plupart des applications de traitement de transaction en ligne (OLTP) doivent être décomposées et réparties dans plusieurs tables logiques lorsqu'elles sont stockées dans un SGBDR.

Par conséquent, une infrastructure de transaction conforme à ACID est nécessaire pour éviter les conditions de concurrence et les problèmes d'intégrité des données pouvant avoir lieu si une application tente de lire un objet qui est en cours d'écriture. Une telle structure de transaction, lorsqu'elle est associée à un schéma relationnel, peut surcharger considérablement le processus d'écriture.

L'implémentation des transactions dans DynamoDB permet d'éviter les problèmes de mise à l'échelle qui sont souvent constatés avec un SGBDR. Pour ce faire, DynamoDB émet une transaction sous la forme d'un appel d'API unique et limite le nombre d'éléments accessibles dans cette même transaction. Les transactions de longue durée peuvent entraîner des problèmes opérationnels en bloquant les données de manière prolongée, voire perpétuelle, car la transaction n'est jamais clôturée.

Pour éviter de tels problèmes dans DynamoDB, les transactions ont été implémentées avec deux opérations d'API distinctes : `TransactWriteItems` et `TransactGetItems`. La sémantique de début et de fin de ces opérations d'API n'est pas commune dans un SGBDR. De plus, DynamoDB impose une limite d'accès à 100 éléments dans une même transaction dans une optique similaire qui est d'empêcher les transactions de longue durée. Pour en savoir plus sur les transactions DynamoDB, consultez Utilisation de transactions.

Ce sont les raisons pour lesquelles, lorsque votre activité exige une réponse à faible latence pour des requêtes à fort trafic, il est généralement judicieux, d'un point de vue technique et économique, de

tirer parti des avantages d'un système NoSQL. Amazon DynamoDB aide à résoudre les problèmes qui restreignent la capacité de mise à l'échelle d'un système relationnel en les évitant.

Les performances d'un SGBDR peinent généralement à évoluer pour les raisons suivantes :

- Elle utilise des jointures coûteuses pour reconstituer les vues de résultats de requête requises.
- Elle normalise les données et les stocke dans plusieurs tables qui nécessitent plusieurs requêtes pour l'écriture sur disque.
- Elle entraîne généralement les coûts de performances d'un système de transaction conforme à ACID.

DynamoDB se met à l'échelle coorectement pour les raisons suivantes :

- Grâce à la flexibilité de son schéma, DynamoDB peut stocker des données hiérarchiques complexes au sein d'un seul élément.
- Une conception de clé composite permet à ce service de stocker les éléments liés proches les uns des autres dans la même table.
- Les transactions sont effectuées dans une même opération. Le nombre d'éléments accessibles est limité à 100 pour éviter les opérations de longue durée.

Les requêtes sur le magasin de données deviennent beaucoup plus simples, en prenant souvent la forme suivante :

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB exécute beaucoup moins de tâches pour renvoyer les données demandées, comparé au SGBDR de l'exemple précédent.

Premiers pas pour la modélisation des données relationnelles dans DynamoDB

> **Important**
>
> Pour concevoir un système NoSQL, il faut un autre état d'esprit que pour un SGBDR. Pour un SGBDR, vous pouvez créer un modèle de données normalisé sans réfléchir aux modèles d'accès. Vous pouvez l'étendre ultérieurement, pour répondre à de nouvelles questions

et de nouveaux besoins d'interrogation. En revanche, dans Amazon DynamoDB, vous ne devez pas commencer à concevoir votre schéma tant que vous ne savez pas à quelle problématique celui-ci doit répondre. Il est absolument essentiel d'identifier au préalable les problèmes métier et les cas d'utilisation de l'application.

Pour commencer à concevoir une table DynamoDB qui pourra être efficacement mise à l'échelle, vous devez d'abord effectuer plusieurs étapes pour identifier les modèles d'accès qui sont requis par les systèmes de support d'exploitation et de support d'activités (OSS/BSS) que celle-ci doit prendre en charge :

- Pour les nouvelles applications, consultez des témoignages d'utilisateurs sur les activités et les objectifs. Documentez les différents cas d'utilisation que vous identifiez et analysez les modèles d'accès dont ceux-ci ont besoin.
- Pour des applications existantes, analysez les journaux de requêtes pour découvrir comment le système est actuellement utilisé et identifier les principaux modèles d'accès.

Après avoir terminé ce processus, vous devriez obtenir une liste ressemblant à ce qui suit :

	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee's phone number(s)
4	Find a customer's phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

Dans une application réelle, votre liste peut être beaucoup plus longue. Mais cette collection représente toute la complexité d'un modèle de requête que vous pourriez trouver dans un environnement de production.

Une approche courante de la conception d'un schéma DynamoDB consiste à identifier les entités de couche d'application, et à utiliser la dénormalisation et le regroupement de clés composites afin de diminuer la complexité des requêtes.

Dans DynamoDB, cela implique d'utiliser des clés de tri composites, des index secondaires globaux surchargés, des tables/index partitionnés et d'autres modèles de conception. Vous pouvez utiliser ces éléments pour structurer les données afin qu'une application puisse récupérer ce dont elle a besoin pour un modèle d'accès donné à l'aide d'une requête unique sur une table ou un index. Le modèle principal que vous pouvez utiliser pour modéliser le schéma normalisé illustré dans Modélisation relationnelle est le modèle de liste adjacente. D'autres modèles utilisés dans cette conception peuvent inclure le partitionnement d'écriture d'index secondaire global, la surcharge d'index secondaire globale, des clés composites et des regroupements matérialisés.

> **Important**
>
> En règle générale, vous devez gérer le moins de tables possible dans une application DynamoDB. Les cas où des données de série chronologique volumineuses sont impliquées ou dans lesquels les ensembles de données ont des modèles d'accès très différents sont des exemples d'exceptions. Une table unique avec des index inversés peut généralement permettre à des requêtes simples de créer et récupérer les structures de données hiérarchiques complexes dont votre application a besoin.

Pour utiliser NoSQL Workbench pour DynamoDB afin de mieux visualiser votre conception de clé de partition, consultez Création de modèles de données avec NoSQL Workbench.

Exemple de modélisation des données relationnelles dans DynamoDB

Cet exemple décrit comment modéliser des données relationnelles dans Amazon DynamoDB. La conception d'une table DynamoDB correspond au schéma de traitement des commandes relationnel illustré dans Modélisation relationnelle. Elle respecte le Modèle de conception de liste adjacente, méthode couramment utilisée pour représenter les structures de données relationnelles dans DynamoDB.

Le modèle de conception nécessite que vous définissiez un ensemble de types d'entités qui coïncident généralement avec les différentes tables du schéma relationnel. Les éléments d'entité sont ensuite ajoutés à la table à l'aide d'une clé primaire composée (partition et tri). La clé de partition de ces éléments d'entité est l'attribut qui identifie de façon unique l'élément et qui est appelé PK de manière générique sur tous les éléments. L'attribut de la clé de tri contient une valeur d'attribut que vous pouvez utiliser pour un index inversé ou un index secondaire global. Il est appelé SK de manière générique.

Vous définissez les entités suivantes, qui prennent en charge le schéma relationnel de saisie des commandes.

1. HR-Employee - PK: EmployeeID, SK: Employee Name
2. HR-Region - PK: RegionID, SK: Region Name
3. HR-Country - PK : CountryId, SK : Nom du pays
4. HR-Location - PK: LocationID, SK: Country Name
5. HR-Job - PK: JobID, SK: Job Title
6. Département des ressources humaines - PK : DepartmentID, SK : DepartmentName
7. OE-Customer - PK : CustomerID, SK : ID AccountRep
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Product Name
10. OE-Warehouse - PK: WarehouseID, SK: Region Name

Une fois ces éléments d'entité ajoutés à la table, vous pouvez définir leurs relations en ajoutant des éléments de bord aux partitions des éléments d'entité. La table suivante illustre cette étape.
Exemple de table illustrant les relations entre des éléments d'entité.

Dans cet exemple, les partitions `Employee`, `Order` et `Product Entity` de la table disposent d'éléments supplémentaires contenant des pointeurs vers d'autres éléments d'entité de la table. Définissez ensuite quelques index secondaires globaux (GSIs) pour prendre en charge tous les modèles d'accès définis précédemment. Les éléments d'entité n'utilisent pas tous le même type de valeur pour l'attribut de la clé primaire ou l'attribut de la clé de tri. La seule obligation est que les attributs de la clé primaire et de la clé de tri présents soient insérés dans la table.

Le fait que certaines de ces entités utilisent des noms propres et d'autres utilisent d'autres entités IDs comme valeurs de clé de tri permet au même index secondaire global de prendre en charge plusieurs types de requêtes. Cette technique est appelée surcharge d'index secondaire global (GSI). Elle permet de dépasser la limite par défaut de 20 index secondaires globaux pour les tables contenant plusieurs types d'éléments. Ceci est illustré par GSI 1 dans le diagramme suivant :
Exemple de table illustrant des index secondaires globaux prenant en charge plusieurs requêtes.

GSI 2 est conçu pour prendre en charge un schéma d'accès aux applications assez courant, qui consiste à obtenir tous les éléments de la table ayant un état spécifique. Dans le cas d'une grande table dont la répartition des éléments est inégale en termes d'états disponibles, ce modèle d'accès

peut entraîner une forte activité, sauf si les éléments sont répartis sur plusieurs partitions logiques pouvant être interrogées en parallèle. Ce modèle de conception s'appelle `write sharding`.

Pour obtenir ceci pour GSI 2, l'application ajoute l'attribut de la clé primaire de GSI 2 à chaque élément Order. Elle le remplit avec un nombre aléatoire compris dans une plage de 0 à N, où N peut être calculé de façon générique à l'aide de la formule suivante, sauf s'il existe une raison spécifique de faire autrement.

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

Supposons par exemple que vous souhaitez obtenir les résultats suivants :

- Jusqu'à 2 millions de commandes dans le système, passant à 3 millions en 5 ans.
- Jusqu'à 20 % de ces commandes seront à l'état OPEN (OUVERT) à un moment donné.
- La taille d'un enregistrement de commande moyen est d'environ 100 octets, avec trois enregistrements `OrderItem` dans la partition de commande de 50 octets chacun, soit une taille d'entité de commande moyenne de 250 octets.

Pour cette table, le calcul du facteur N se présenterait comme suit :

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

Dans ce cas, vous devez répartir toutes les commandes sur au moins 13 partitions logiques sur GSI 2 afin de garantir que la lecture de tous les éléments `Order` dont l'état est `OPEN` n'entraîne pas de surchauffe de la partition sur la couche du stockage physique. Il est recommandé d'augmenter ce nombre en prévision d'éventuelles anomalies dans l'ensemble de données. Il est donc préférable d'utiliser `N = 15`. Comme mentionné précédemment, vous le faites en ajoutant la valeur aléatoire de 0 à N à l'attribut PK GSI 2 de chaque registre `Order` et `OrderItem` inséré sur la table.

Cette répartition suppose que le modèle d'accès qui nécessite le rassemblement de toutes les factures dont l'état est `OPEN` a lieu de manière occasionnelle, afin que vous puissiez utiliser la

capacité de transmission en mode rafale pour satisfaire la demande. Vous pouvez interroger l'index secondaire global suivant à l'aide d'une condition de clé de tri `State` et `Date Range` pour générer un sous-ensemble ou l'ensemble des `Orders` d'un état donné.
Exemple de table illustrant une clé primaire GSI 2 et les attributs projetés.

Dans cet exemple, les éléments sont répartis de manière aléatoire sur 15 partitions logiques. Cette structure fonctionne, car le modèle d'accès nécessite qu'un grand nombre d'éléments soient récupérés. Ainsi, il est peu probable qu'aucun des 15 threads renvoie des jeux de résultats vides, ce qui pourrait représenter un gaspillage de capacité. Une requête utilise toujours une unité de capacité de lecture (RCU, read capacity unit) ou une unité de capacité d'écriture (WCU, write capacity unit), même si rien n'est renvoyé ou si aucune donnée n'est écrite.

Si le modèle d'accès nécessite une requête très rapide sur cet index secondaire global qui renvoie un ensemble de résultats fragmenté, il est probablement préférable d'utiliser un algorithme de hachage pour répartir les éléments plutôt qu'un modèle aléatoire. Dans ce cas, vous pouvez sélectionner un attribut connu lorsque la requête est exécutée au moment du runtime, et hacher cet attribut dans un espace de clé 0 à 14 lors de l'insertion des éléments. Ils peuvent ainsi être lus de manière efficace à partir de l'index secondaire global.

Enfin, vous pouvez réexaminer les modèles d'accès définis précédemment. Voici la liste des modèles d'accès et des conditions de requête à utiliser avec la nouvelle version DynamoDB de l'application pour les accueillir.

Nº de série	Modèles d'accès	Conditions de requête
1	Rechercher les détails des employés par ID d'employé	Clé primaire sur la table, ID="HR-EMPLOYEE »
2	Interroger les détails des employés par nom d'employé	Utiliser GSI-1, PK="Nom de l'employé »
3	Obtenir uniquement les détails de la tâche actuelle d'un employé	Clé primaire sur la table, PK=HR-EMPLOYEE-1, SK commence par « JH »
4	Obtenir des commandes pour un client pour une plage de dates	Utilisez GSI-1, PK=, SK="STATUS-DATE »CUSTOMER1, pour chaque StatusCode

Nº de série	Modèles d'accès	Conditions de requête
5	Afficher toutes les commandes en état OUVERT pour une plage de dates pour tous les clients	Utilisez GSI-2, PK=Query en parallel pour la plage [0..N], SK entre OpenDate1 et OpenDate2
6	Tous les employés embauchés récemment	Utilisez GSI-1, PK="HR-CONFIDENTIAL", SK > date1
7	Trouver tous les employés dans un entrepôt spécifique	Utilisez GSI-1, PK= WAREHOUSE1
8	Obtenir tous les articles de commande d'un produit, y compris les stocks d'entrepôts	Utilisez GSI-1, PK= PRODUCT1
9	Obtenez des clients par représentant du compte	Utiliser GSI-1, PK=ACCOUNT-REP
10	Obtenir des commandes par représentant du compte et par date	Utilisez GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE », pour chaque StatusCode
11	Obtenir tous les employés ayant un intitulé de poste spécifique	Utiliser GSI-1, PK=JOBTITLE
12	Obtenir le stock par produit et par entrepôt	Clé primaire sur la table, PK=OE-PRODUCT1, SK= PRODUCT1
13	Obtenir l'inventaire total des produits	Clé primaire sur la table, PK=OE-PRODUCT1, SK= PRODUCT1

Nº de série	Modèles d'accès	Conditions de requête
14	Obtenir le classement des représentants du compte par nombre total de commandes et par période de vente	Utilisez GSI-1, PK=YYYY-Q1, =False scanIndexForward

Migration à partir d'une base de données relationnelle vers DynamoDB

La migration d'une base de données relationnelle vers DynamoDB nécessite une planification minutieuse pour garantir un résultat réussi. Ce guide vous aidera à comprendre le fonctionnement de ce processus, les outils dont vous disposez, puis comment évaluer les stratégies de migration potentielles et sélectionner celle qui répondra à vos besoins.

Rubriques

- Raisons justifiant la migration vers DynamoDB
- Considérations relatives à la migration d'une base de données relationnelle vers DynamoDB
- Comprendre le fonctionnement d'une migration vers DynamoDB
- Outils permettant de migrer vers DynamoDB
- Choix de la stratégie appropriée pour migrer vers DynamoDB
- Réalisation d'une migration hors ligne vers DynamoDB
- Exécution d'une migration hybride vers DynamoDB
- Réalisation d'une migration en ligne vers DynamoDB en faisant migrer chaque table 1:1
- Réalisation d'une migration en ligne vers DynamoDB à l'aide d'une table intermédiaire personnalisée

Raisons justifiant la migration vers DynamoDB

La migration vers Amazon DynamoDB présente de nombreux avantages intéressants pour les entreprises et les organisations. Voici quelques avantages clés qui font de DynamoDB un choix intéressant pour la migration de bases de données :

- Capacité de mise à l'échelle : DynamoDB est conçu pour gérer des charges de travail massives et s'adapter facilement à l'augmentation des volumes de données et du trafic. Avec DynamoDB, vous pouvez facilement augmenter ou réduire verticalement votre base de données en fonction de la demande, afin que vos applications puissent gérer des pics de trafic soudains sans compromettre les performances.
- Performances : DynamoDB offre un accès aux données à faible latence, ce qui permet aux applications de récupérer et de traiter les données à une vitesse exceptionnelle. Son architecture

distribuée garantit que les opérations de lecture et d'écriture sont réparties sur plusieurs nœuds, et offre des temps de réponse constants de l'ordre de la milliseconde, même à des taux de demandes élevés.

- Entièrement géré : DynamoDB est un service entièrement géré fourni par AWS. Cela signifie qu'il AWS gère les aspects opérationnels de la gestion des bases de données, notamment le provisionnement, la configuration, les correctifs, les sauvegardes et le dimensionnement. Cela vous permet de vous concentrer davantage sur le développement de vos applications et moins sur les tâches d'administration de base de données.
- Architecture sans serveur : DynamoDB prend en charge un modèle sans serveur, appelé DynamoDB à la demande, dans lequel vous ne payez que pour les demandes de lecture et d'écriture réellement effectuées par votre application, sans qu'aucun provisionnement de capacité initial ne soit requis. Ce pay-per-request modèle offre une rentabilité et une charge opérationnelle minimale, car vous ne payez que pour les ressources que vous consommez, sans qu'il soit nécessaire de provisionner et de surveiller les capacités.
- Flexibilité NoSQL : contrairement aux bases de données relationnelles traditionnelles, DynamoDB suit un modèle de données NoSQL, offrant ainsi de la flexibilité dans la conception des schémas. Avec DynamoDB, vous pouvez stocker des données structurées, semi-structurées et non structurées, ce qui le rend parfaitement adapté à la gestion de types de données divers et évolutifs. Cette flexibilité permet d'accélérer les cycles de développement et de s'adapter plus facilement à l'évolution des besoins de l'entreprise.
- Haute disponibilité et durabilité : DynamoDB réplique les données dans plusieurs zones de disponibilité au sein d'une région, garantissant ainsi une haute disponibilité et une durabilité des données. Il gère automatiquement la réplication, le basculement et la restauration, minimisant ainsi le risque de perte de données ou d'interruption de service. DynamoDB fournit un SLA de disponibilité allant jusqu'à 99,999 %.
- Sécurité et conformité : DynamoDB s'intègre à Gestion des identités et des accès AWS pour le contrôle précis des accès. Il permet un chiffrement au repos et en transit, garantissant ainsi la sécurité de vos données. DynamoDB respecte également différentes normes de conformité (HIPAA, PCI DSS et RGPD, par exemple), ce qui vous permet de répondre aux exigences réglementaires.
- Intégration à l' AWS écosystème : dans le cadre de l' AWS écosystème, DynamoDB s'intègre parfaitement à d' AWS autres services, AWS Lambda tels que CloudFormation, et. AWS AppSync Cette intégration vous permet de créer des architectures sans serveur, de tirer parti de l'infrastructure sous forme de code et de créer des applications pilotées par les données en temps réel.

Considérations relatives à la migration d'une base de données relationnelle vers DynamoDB

Les systèmes de gestion de bases de données relationnelles et les bases de données NoSQL ont chacun leurs forces et leurs faiblesses. Ces différences entraînent une conception des bases de données très différente d'un système à l'autre :

Type de tâche	Base de données relationnelle	Base de données NoSQL
Interrogation de la base de données	Dans les bases de données relationnelle, les données peuvent être interrogées avec souplesse, mais les requêtes sont relativement coûteuses et ne sont pas bien mises à l'échelle dans les situations de trafic intense (consultez Premiers pas pour la modélisation des données relationnelles dans DynamoDB). Une applicati on de base de données relationnelle peut implément er une logique métier dans les procédures stockées, les sous-requêtes SQL, les requêtes de mise à jour en masse et les requêtes d'agrégation.	Dans une base de données NoSQL comme DynamoDB, les données peuvent être interrogées efficacement d'un nombre limité de façons, en dehors desquelles les requêtes peuvent être coûteuses et lentes. Les écritures dans DynamoDB sont des singletons. La logique métier des applicati ons qui s'exécutait auparavant dans des procédures stockées doit être refactorisée pour s'exécuter en dehors de DynamoDB dans du code personnalisé exécuté sur un hôte tel qu'Amazon ou. EC2 AWS Lambda
Conception de la base de données	La conception se concentre avant tout sur la flexibilité, sans se préoccuper des détails de l'implémentation ni des performances. En général, l'optimisation des requêtes	Votre schéma est conçu spécifiquement pour que les requêtes les plus courantes et les plus importantes soient aussi rapides et économiques que possible. Les structures

Type de tâche	Base de données relationnelle	Base de données NoSQL
	n'affecte pas la conception du schéma, mais la normalisation est très importante.	de vos données sont adaptées aux exigences spécifiques de vos cas d'utilisation.

La conception d'une base de données NoSQL nécessite un état d'esprit différent de celui de la conception d'un système de gestion de base de données relationnelle (SGBDR). Pour un SGBDR, vous pouvez créer un modèle de données normalisé sans réfléchir aux modèles d'accès. Vous pouvez l'étendre ultérieurement, pour répondre à de nouvelles questions et de nouveaux besoins d'interrogation. Vous pouvez organiser chaque type de données dans sa propre table.

Avec la conception NoSQL, vous pouvez concevoir votre schéma pour DynamoDB lorsque vous savez à quelle problématique celui-ci doit répondre. Il est essentiel d'identifier au préalable les problèmes métier et les modèles de lecture et d'écriture de l'application. Vous devez aussi essayer de gérer le moins de tables possible dans une application DynamoDB. Le fait d'avoir moins de tables rend les choses plus évolutives, nécessite moins de gestion des autorisations et réduit les frais généraux pour votre application DynamoDB. Cela peut également contribuer à maintenir des coûts de sauvegarde globalement plus faibles.

La tâche de modélisation des données relationnelles pour DynamoDB et la création d'une nouvelle version de l'application frontale font l'objet d'une rubrique distincte. Ce guide part du principe que vous disposez d'une nouvelle version de votre application conçue pour utiliser DynamoDB, mais vous devez tout de même déterminer la meilleure façon de migrer et de synchroniser les données historiques lors du basculement.

Considérations sur le dimensionnement

La taille maximale de chaque élément (rangée) que vous stockez dans une table DynamoDB est de 400 Ko. Pour de plus amples informations, veuillez consulter the section called "Quotas". La taille de l'élément est déterminée par la taille totale de tous les noms d'attributs et de toutes les valeurs d'attribut d'un élément. Pour de plus amples informations, veuillez consulter the section called "Tailles et formats d'élément".

Si votre application a besoin de stocker dans un élément davantage de données que ne le permet la limite de taille dans DynamoDB, scindez l'élément en une collection d'éléments, compressez les données de l'élément, ou stockez l'élément en tant qu'objet dans Amazon Simple Storage Service (Amazon S3), tout en stockant l'identifiant d'objet Amazon S3 dans votre élément DynamoDB.

Consultez the section called “Éléments volumineux”. Le coût de mise à jour d’un élément dépend de sa taille complète. Pour les charges de travail qui nécessitent des mises à jour fréquentes des éléments existants, la mise à jour de petits éléments d’un ou deux Ko coûtera moins cher que celle d’éléments plus volumineux. Pour plus d’informations sur les collections d’éléments, consultez the section called “Travailler avec les collections d'éléments”.

Lorsque vous choisissez les attributs de partition et de clé de tri, les autres paramètres de table, la taille et la structure des éléments, et si vous souhaitez créer des index secondaires, n’oubliez pas de consulter la documentation relative à la modélisation DynamoDB ainsi que le guide d’the section called “Optimisation des coûts”. Assurez-vous de tester votre plan de migration afin que votre solution DynamoDB soit rentable et réponde aux fonctionnalités et limites de DynamoDB.

Comprendre le fonctionnement d’une migration vers DynamoDB

Avant de passer en revue les outils de migration mis à notre disposition, réfléchissez à la manière dont les écritures sont traitées par DynamoDB.

L’opération d’écriture par défaut et la plus courante est une opération d’API `PutItem` unique. Vous pouvez effectuer une opération `PutItem` en boucle pour traiter des jeux de données. DynamoDB prend en charge un nombre pratiquement illimité de connexions simultanées. Par conséquent, en supposant que vous puissiez configurer et exécuter une routine de chargement massivement multithread telle MapReduce que ou Spark, la vitesse d'écriture n'est limitée que par la capacité de la table cible (qui est également généralement illimitée).

Lorsque vous chargez des données dans DynamoDB, il est important de comprendre la vitesse d’écriture de votre chargeur. Si les éléments (rangées) que vous chargez ont une taille inférieure ou égale à 1 Ko, cette vélocité correspond simplement au nombre d’éléments par seconde. La table cible peut ensuite être provisionnée avec suffisamment de WCU (unités de capacité d’écriture) pour gérer ce taux. Si votre chargeur dépasse la capacité provisionnée à chaque seconde, les demandes supplémentaires peuvent être limitées ou rejetées. Vous pouvez vérifier les limites dans les CloudWatch graphiques de l'onglet de surveillance de la console DynamoDB.

La deuxième opération qui peut être effectuée est avec une API associée appelée `BatchWriteItem`. `BatchWriteItem` vous permet de combiner jusqu’à 25 demandes d’écriture en un seul appel d’API. Elles sont reçues par le service et traitées comme des demandes `PutItem` distinctes adressées à la table. Actuellement, lors du choix`BatchWriteItem`, vous ne bénéficierez pas de l'avantage des nouvelles tentatives automatiques incluses dans le AWS SDK lorsque vous

passez des appels singleton avec. `PutItem` Ainsi, s'il y a des erreurs (telles que des exceptions de limitation), vous devrez rechercher la liste des écritures ayant échoué dans l'appel de réponse à `BatchWriteItem`. Pour plus d'informations sur la gestion des avertissements de limitation au cas où ils seraient détectés dans les graphiques de CloudWatch régulation, voir. the section called "étranglement"

Le troisième type d'importation de données est possible grâce à la fonctionnalité DynamoDB Import from S3. Cette fonctionnalité vous permet de créer un jeu de données volumineux dans Amazon S3 et de demander à DynamoDB d'importer automatiquement les données dans une nouvelle table. L'importation n'est pas instantanée et prend un temps proportionnel à la taille du jeu de données. Cependant, cela est pratique car aucune plateforme ETL ni aucun code DynamoDB personnalisé n'est nécessaire. DynamoDB charge les données dans une nouvelle table créée par l'importation. Actuellement, cela ne vous permet pas de charger des données dans une table existante. DynamoDB importe les données telles quelles, sans aucune transformation. De même que pour `PutItem`, aucun un processus en amont n'est nécessaire et les données sont écrites dans le format de votre choix dans un compartiment Amazon S3.

Outils permettant de migrer vers DynamoDB

Il existe plusieurs outils de migration et ETL courants que vous pouvez utiliser pour migrer des données vers DynamoDB.

Amazon fournit une multitude d'outils de données qui peuvent être utilisés pour la migration, notamment AWS Database Migration Service (DMS), AWS Glue, Amazon EMR et Amazon Managed Streaming for Apache Kafka. Tous ces outils peuvent être utilisés pour effectuer une migration pendant une durée d'indisponibilité, et ils peuvent tirer parti des fonctionnalités de capture des données modifiées (CDC) des bases de données relationnelles pour prendre en charge les migrations en ligne. Lorsque vous choisissez un outil, il est utile de prendre en compte les compétences et l'expérience de votre organisation avec chaque outil, ainsi que les fonctionnalités, les performances et le coût de chacun d'entre eux.

De nombreux clients choisissent d'écrire leurs propres scripts et tâches de migration afin de créer des transformations de données personnalisées pour le processus de migration. Si vous envisagez d'exploiter une table DynamoDB à volume élevé avec un trafic d'écriture important ou des tâches régulières de chargement en masse de grande taille, vous souhaiterez peut-être écrire vous-même le code de migration afin de vous familiariser avec le comportement de DynamoDB en cas de trafic d'écriture élevé. Des scénarios tels que la gestion des limitations et le provisionnement efficace des tables peuvent être expérimentés au début du projet lors d'une migration pratique.

Choix de la stratégie appropriée pour migrer vers DynamoDB

Une application de base de données relationnelle de grande taille peut s'étendre sur une centaine de tables ou plus et prendre en charge plusieurs fonctions d'application différentes. Lorsque vous vous apprêtez à effectuer une migration de grande envergure, pensez à diviser votre application en composants plus petits ou en microservices, et à migrer un petit ensemble de tables à la fois. Vous pouvez ensuite migrer des composants supplémentaires vers DynamoDB par vagues.

Lors du choix d'une stratégie de migration, différents facteurs peuvent vous orienter vers une solution ou une autre. Nous pouvons présenter ces options dans un arbre de décision afin de simplifier les options qui s'offrent à nous en fonction de nos besoins et des ressources disponibles. Les concepts sont brièvement mentionnés ici (mais seront abordés plus en détail plus loin dans le guide) :

- Migration hors ligne : si votre application peut tolérer une certaine durée d'indisponibilité pendant la migration, cela simplifiera le processus de migration.
- Migration hybride : cette approche permet une durée de fonctionnement partielle pendant une migration, par exemple en autorisant les lectures mais pas les écritures, ou en autorisant les lectures et les insertions mais pas les mises à jour et les suppressions.
- Migration en ligne : les applications qui ne tolèrent aucune durée d'indisponibilité pendant la migration sont moins faciles à migrer et peuvent nécessiter une planification importante et un développement personnalisé. L'une des décisions clés consiste à estimer et à évaluer les coûts liés à la mise en place d'un processus de migration personnalisé par rapport au coût pour l'entreprise d'une durée d'indisponibilité pendant le basculement.

If	Et	Then
Vous pouvez désactiver l'application pendant un certain temps pendant une période de maintenance pour effectuer la migration des données. Il s'agit d'une migration hors ligne.		Utilisez AWS DMS et effectuez une migration hors ligne à l'aide d'une tâche de chargement complet. Préformez les données source à l'aide d'un code SQL `VIEW` si vous le souhaitez.
Vous pouvez exécuter l'application en mode lecture		Désactivez les écritures dans l'application ou la base de

If	Et	Then
seule pendant la migration. Il s’agit d’une migration hybride.		données source. Utilisez AWS DMS et effectuez une migration hors ligne à l'aide d'une tâche de chargement complet.
Vous pouvez exécuter l’application avec des lectures et des insertions de nouveaux enregistrements, mais pas de mises à jour ni de suppressi ons pendant la migration. Il s’agit d’une migration hybride.	Vous avez des compétences en développement d’applica tions et pouvez mettre à jour l’application relationnelle existante pour effectuer des écritures doubles, y compris dans DynamoDB, pour tous les nouveaux enregistrements	Utilisez AWS DMS et effectuez une migration hors ligne à l'aide d'une tâche de chargement complet. Déployez simultanément une version de l’applica tion existante qui autorise les lectures et effectue des écritures doubles.
Vous avez besoin d’une migration avec une durée d’indisponibilité minimale. Il s’agit d’une migration en ligne.	• Vous migrez les tables sources 1-vers-1 dans DynamoDB sans modificat ions majeures du schéma.	AWS DMS À utiliser pour effectuer une migration de données en ligne. Exécutez une tâche de chargement en bloc suivie d’une tâche de synchronisation CDC.
Vous avez besoin d’une migration avec une durée d’indisponibilité minimale. Il s’agit d’une migration en ligne.	• Vous combinez des tables sources en un nombre réduit de tables DynamoDB selon le schéma empilé ou la philosophie de la table unique. • Vous avez des compétences en développement de bases de données dorsales et des capacités inutilisées sur l’hôte SQL.	Créez la table NoSQL Ready dans la base de données SQL. Remplissez-le et synchronisez-le à l'aide de déclencheurs JOINs UNIONs VIEWs, de procédures stockées.

If	Et	Then
Vous avez besoin d'une migration avec une durée d'indisponibilité minimale. Il s'agit d'une migration en ligne.	• Vous combinez des tables sources en un nombre réduit de tables DynamoDB selon la philosophie de la table unique. Par exemple : • Vous n'avez pas des compétences en développement de bases de données dorsales et des capacités inutilisées sur l'hôte SQL.	Envisagez les approches de migration hybrides ou hors ligne.
Vous avez besoin d'une migration avec une durée d'indisponibilité minimale. Il s'agit d'une migration en ligne.	Vous pouvez ignorer la migration des données de transaction historiques ou les archiver dans Amazon S3 au lieu de les migrer. Il suffit de migrer quelques petites tables statiques.	Écrivez un script ou utilisez un outil ETL pour migrer les tables. Préformez les données source à l'aide d'un code SQL `VIEW` si vous le souhaitez.

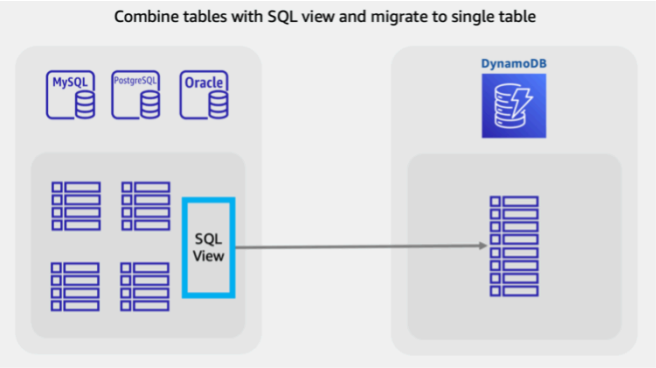
Réalisation d'une migration hors ligne vers DynamoDB

Les migrations hors ligne sont adaptées lorsque vous pouvez autoriser une durée d'indisponibilité pour effectuer la migration. Les bases de données relationnelles prennent généralement au moins une certaine durée d'indisponibilité arrêt par mois pour des raisons de maintenance et de correction, lesquelle peuvent être plus longues pour les mises à niveau matérielles ou les mises à niveau de versions majeures.

Amazon S3 peut être utilisé comme zone intermédiaire lors d'une migration. Les données stockées au format CSV (valeurs séparées par des virgules) ou DynamoDB JSON peuvent être automatiquement importées dans une nouvelle table DynamoDB à l'aide de la Fonctionnalité d'importation DynamoDB à partir de S3.

Vous souhaiterez peut-être combiner des tables pour tirer parti de modèles d'accès NoSQL uniques (par exemple, transformer quatre tables héritées en une seule table DynamoDB). Une demande

de document contenant une valeur clé unique ou une requête pour une collection d'éléments pré-groupés est généralement renvoyée avec une latence supérieure à celle d'une base de données SQL qui effectue une jointure à plusieurs tables. Cela complique toutefois la tâche de migration. Une vue SQL peut effectuer le travail au sein de la base de données source pour préparer un jeu de données unique représentant les quatre tables d'un seul ensemble.



Cette vue permet de dénormaliser les tables `JOIN` ou de maintenir les entités normalisées et d'empiler les tables à l'aide d'une instruction SQL `UNION`. Les principales décisions relatives à la refonte des données relationnelles sont abordées dans cette vidéo. Pour les migrations hors ligne, l'utilisation d'une vue pour combiner des tables est un excellent moyen de façonner les données d'un schéma de table unique DynamoDB.

Planifier

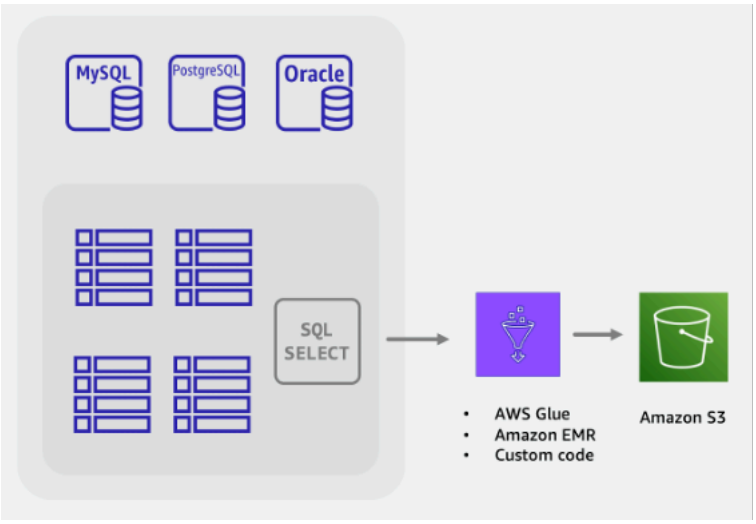
Exécution d'une migration hors ligne à l'aide d'Amazon S3

Outils

- Tâche ETL pour extraire et transformer des données SQL et les stocker dans un compartiment S3 tel que :
  - AWS Database Migration Service, service capable de charger des données historiques en bloc et de traiter les enregistrements CDC pour synchroniser les tables source et cible.
  - AWS Glue
  - Amazon EMR
  - Votre propre code personnalisé
- Fonctionnalité d'importation DynamoDB à partir de S3

Étapes de migration hors ligne :

1. Créez une tâche ETL capable d'interroger la base de données SQL, de transformer les données des tables au format DynamoDB JSON ou CSV et de les enregistrer dans un compartiment S3.



2. La fonctionnalité d'importation DynamoDB à partir de S3 est invoquée pour créer une nouvelle table et charger automatiquement les données depuis votre compartiment S3.

La migration entièrement hors ligne est simple et directe, mais elle risque de ne pas être populaire auprès des propriétaires et des utilisateurs d'applications. Les utilisateurs en tireraient bénéfice si l'application pouvait fournir des niveaux de service réduits pendant la migration, au lieu de ne pas fournir de service du tout.

Vous pouvez ajouter une fonctionnalité pour désactiver les écritures pendant la migration hors ligne, tout en permettant aux lectures de se poursuivre normalement. Les utilisateurs de l'application peuvent toujours parcourir et interroger en toute sécurité les données existantes pendant la migration des données relationnelles. Si c'est ce que vous recherchez, poursuivez votre lecture pour en savoir plus sur les migrations hybrides.

Exécution d'une migration hybride vers DynamoDB

Bien que toutes les applications de base de données exécutent des opérations de lecture et d'écriture, les types d'opérations d'écriture effectués doivent être pris en compte lors de la planification d'une migration hybride ou en ligne. Les écritures de base de données peuvent être classées en trois compartiments : insertions, mises à jour et suppressions. Certaines applications peuvent ne pas nécessiter le traitement immédiat des suppressions. Ces applications peuvent, par exemple, reporter les suppressions à un processus de nettoyage en bloc à la fin du mois. Ces types

d’applications peuvent être plus simples à migrer tout en garantissant une durée de fonctionnement partielle.

Planifier

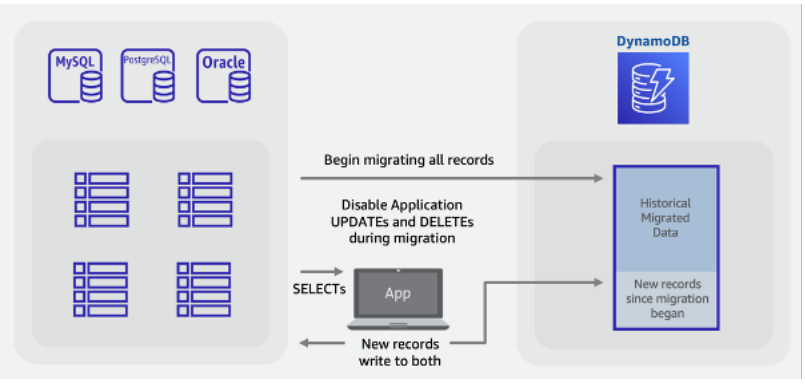
Effectuez une online/offline migration hybride avec deux écritures d'application

Outils

- Tâche ETL pour extraire et transformer des données SQL et les stocker dans un compartiment S3 tel que :
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Votre propre code personnalisé

Étapes de migration hybride :

1. Créez la table DynamoDB cible. Ce tableau recevra à la fois des données historiques en bloc et de nouvelles données en temps réel
2. Créez une version de l’application héritée dont les suppressions et les mises à jour sont désactivées tout en effectuant toutes les insertions sous forme de double écriture dans la base de données SQL et DynamoDB
3. Commencez le travail ou la AWS DMS tâche ETL pour compléter les données existantes et déployer la nouvelle version de l'application en même temps
4. Une fois le travail de remplissage terminé, DynamoDB disposera de tous les enregistrements existants et nouveaux et sera prêt pour le basculement des applications



> **Note**
>
> La tâche de remplissage écrit directement depuis SQL vers DynamoDB. Nous ne pouvons pas utiliser la fonctionnalité d'importation S3 comme dans l'exemple de migration hors ligne, car cette fonctionnalité crée une nouvelle table qui ne sera active qu'après le chargement des données par DynamoDB.

Réalisation d'une migration en ligne vers DynamoDB en faisant migrer chaque table 1:1

De nombreuses bases de données relationnelles disposent d'une fonctionnalité appelée Change Data Capture (CDC), qui permet aux utilisateurs de demander une liste des modifications apportées à une table avant ou après un instant dans le passé. CDC utilise des journaux internes pour activer cette fonctionnalité et il n'est pas nécessaire que la table contienne une colonne d'horodatage pour fonctionner.

Lorsque vous migrez un schéma de tables SQL vers une base de données NoSQL, vous pouvez si vous le souhaitez combiner et remodeler vos données en un nombre réduit de tables. Cela vous permettra de collecter des données en un seul endroit et d'éviter d'avoir à joindre manuellement les données associées lors d'opérations de lecture en plusieurs étapes. Cependant, la mise en forme des données d'une seule table n'est pas toujours requise et vous devez parfois migrer des tables 1-vers-1 dans DynamoDB. Ces migrations de tables 1-vers-1 individuelles sont moins compliquées car vous pouvez tirer parti de la fonctionnalité CDC de la base de données source, en utilisant des outils ETL courants qui prennent en charge ce type de migration. Les données de chaque ligne peuvent toujours être transformées dans de nouveaux formats, mais la portée de chaque table reste la même.

Envisagez de migrer les tables SQL 1-vers-1 dans DynamoDB, tout en gardant à l'esprit que DynamoDB ne prend pas en charge les jointures côté serveur. Vous devez ajouter une logique à votre application pour combiner les données de plusieurs tables.

Planifier

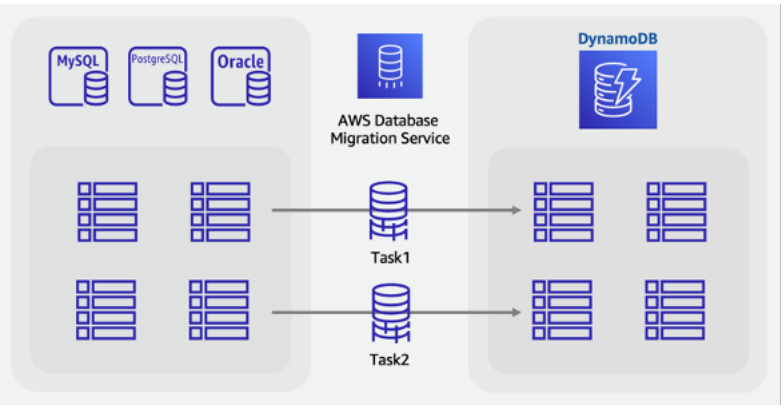
Effectuez une migration en ligne de chaque table dans DynamoDB à l'aide de AWS DMS

Outils

- AWS DMS (DMS)

Étapes de migration en ligne :

1. Identifiez les tables de votre schéma source qui seront migrées
2. Créez le même nombre de tables dans DynamoDB avec la même structure de clés que dans la source
3. Création d'un serveur de réplication AWS DMS et configuration des points de terminaison source et cible
4. Définissez toutes les transformations requises par ligne (telles que les colonnes concaténées ou la conversion des dates au format de chaîne ISO-8601)
5. Créez une tâche de migration pour chaque table pour le chargement complet et la capture des données de modification
6. Surveillez ces tâches jusqu'au début de la phase de réplication en cours
7. À ce stade, vous pouvez effectuer des audits de validation, puis transférer les utilisateurs vers l'application qui lit et écrit dans DynamoDB



Réalisation d'une migration en ligne vers DynamoDB à l'aide d'une table intermédiaire personnalisée

Comme dans le scénario de migration hors ligne ci-dessus, vous pouvez choisir de combiner des tables pour tirer parti de modèles d'accès NoSQL uniques (par exemple, transformer quatre tables héritées en une seule table DynamoDB). Une instruction SQL `VIEW` peut effectuer le travail au sein de la base de données source pour préparer un jeu de données unique représentant les quatre tables d'un seul ensemble.

Toutefois, pour les migrations en ligne avec des données en temps réel et changeantes, vous ne pouvez pas tirer parti des fonctionnalités de CDC car elles ne sont pas prises en charge pour les

VIEWs. Si vos tables incluent une colonne d'horodatage mise à jour pour la dernière fois et que celle-ci y est incorporée dans la VIEW, vous pouvez créer une tâche ETL personnalisée qui les utilise pour effectuer un chargement en bloc avec synchronisation.

Une nouvelle approche pour relever ce défi consiste à utiliser des fonctionnalités SQL standard, telles que les vues, les procédures stockées et les déclencheurs pour créer une nouvelle table SQL au format DynamoDB NoSQL final souhaité.

Si votre serveur de base de données dispose de la capacité inutilisée, il est possible de créer cette table intermédiaire unique avant le début de la migration. Pour ce faire, vous devez écrire une procédure stockée qui lira les données des tables existantes, transformera les données selon les besoins et écrira dans la nouvelle table intermédiaire. Vous pouvez ajouter un ensemble de déclencheurs pour répliquer les modifications apportées aux tables dans la table intermédiaire en temps réel. Si les déclencheurs ne sont pas autorisés conformément à la politique de l'entreprise, les modifications apportées aux procédures stockées peuvent produire le même résultat. Vous devez ajouter quelques lignes de code à toute procédure qui écrit des données, afin d'écrire en outre les mêmes modifications dans la table intermédiaire.

La mise en place de cette table intermédiaire entièrement synchronisée avec les tables héritées d'applications constitue un excellent point de départ pour une migration en direct. Les outils utilisant la base de données CDC pour effectuer des migrations dynamiques, tels que AWS DMS, peuvent désormais être utilisés par rapport à cette table. L'avantage de cette approche est qu'elle utilise des compétences SQL bien connues et des fonctionnalités disponibles dans le moteur de base de données relationnelle.

Planifier

Effectuez une migration en ligne à l'aide d'une table intermédiaire SQL en utilisant AWS DMS

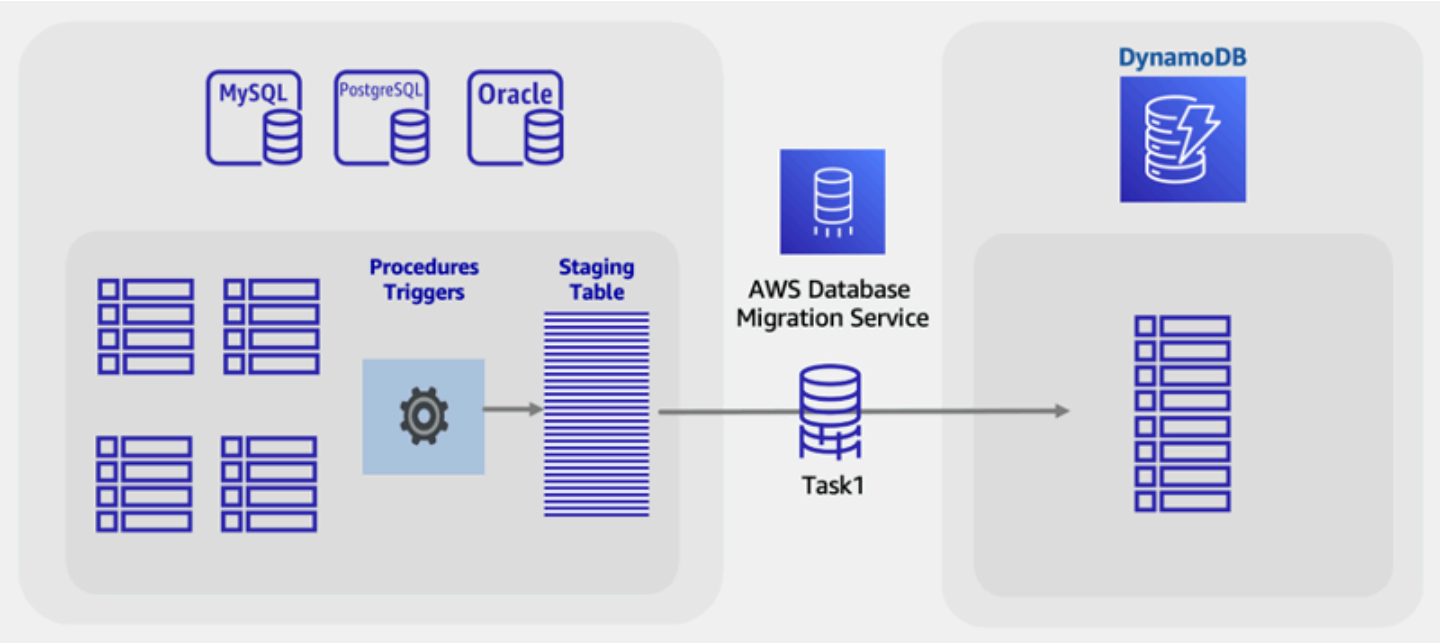
Outils

- Procédures stockées ou déclencheurs SQL personnalisés
- AWS DMS

Étapes de migration en ligne :

1. Dans le moteur de base de données relationnelle source, assurez-vous de disposer d'un espace disque et d'une capacité de traitement supplémentaires.

2. Créez une nouvelle table intermédiaire dans la base de données SQL, avec les horodatages ou les fonctionnalités CDC activées
3. Écriture et exécution d'une procédure stockée pour copier les données d'une table relationnelle existante dans la table intermédiaire
4. Déployez des déclencheurs ou modifiez les procédures existantes pour effectuer une double écriture dans la nouvelle table intermédiaire tout en effectuant des écritures normales dans les tables existantes
5. Exécutez AWS DMS pour migrer et synchroniser cette table source vers une table DynamoDB cible



Ce guide présentait plusieurs considérations et approches relatives à la migration des données de base de données relationnelle vers DynamoDB, en mettant l'accent sur la réduction des durées d'indisponibilité et l'utilisation d'outils et de techniques de base de données courants. Pour plus d'informations, consultez les ressources suivantes :

- AWS DMS Guide de l'utilisateur
- AWS Glue Guide de l'utilisateur
- Bonnes pratiques pour la migration d'un SGBDR vers DynamoDB

NoSQL Workbench pour DynamoDB

NoSQL Workbench pour Amazon DynamoDB est une application côté client multiplateforme pour le développement et les opérations de base de données moderne. Il est disponible pour Windows, macOS et Linux. NoSQL Workbench est un outil de développement visuel qui propose des fonctionnalités de modélisation de données, de visualisation de données et de développement de requêtes, afin de vous aider à concevoir, créer, interroger et gérer des tables DynamoDB. NoSQL Workbench inclut désormais DynamoDB local en tant que composant facultatif du processus d'installation, ce qui facilite la modélisation des données dans DynamoDB local. Pour en savoir plus sur DynamoDB local et ses exigences, consultez Configuration de DynamoDB Local (version téléchargeable) .

Modélisation des données

NoSQL Workbench pour DynamoDB vous permet de créer des modèles de données ou de concevoir des modèles à partir de modèles de données existants, qui pourront satisfaire les modèles d'accès aux données de votre application. À la fin du processus, vous pouvez également importer et exporter le modèle de données conçu. Pour plus d'informations, consultez Création de modèles de données avec NoSQL Workbench.

Visualisation des données

Le visualiseur du modèle de données fournit un canevas sur lequel vous pouvez mapper des requêtes et visualiser les modèles d'accès (facettes) de l'application sans avoir à écrire de code. Chaque facette correspond à un modèle d'accès différent dans DynamoDB. Vous pouvez générer automatiquement des données d'exemple à utiliser dans votre modèle de données. Pour plus d'informations, consultez Visualisation des modèles d'accès de données.

Création des opérations

NoSQL Workbench fournit une interface utilisateur graphique enrichie pour développer et tester des requêtes. Vous pouvez utiliser le créateur d'opérations pour consulter, explorer et interroger des ensembles de données en temps réel. Le créateur d'opérations structurées prend en charge l'expression de projection ainsi que l'expression de condition et génère des exemples de code dans plusieurs langages. Vous pouvez cloner directement des tables d'un compte Amazon DynamoDB vers un autre dans différentes régions. Vous pouvez également cloner directement des tables entre DynamoDB local et un compte Amazon DynamoDB pour copier plus rapidement le schéma de clés de votre table (et éventuellement le schéma GSI et les éléments) entre vos

environnements de développement. Pour plus d'informations, consultez Exploration des jeux de données et des opérations de création avec NoSQL Workbench.

La vidéo ci-dessous détaille les concepts de modélisation des données avec NoSQL Workbench.

Rubriques

- Télécharger NoSQL Workbench pour DynamoDB
- Installer NoSQL Workbench pour DynamoDB
- Création de modèles de données avec NoSQL Workbench
- Visualisation des modèles d'accès de données
- Exploration des jeux de données et des opérations de création avec NoSQL Workbench
- Exemples de modèle de données pour NoSQL Workbench
- Historique de version pour NoSQL Workbench

Télécharger NoSQL Workbench pour DynamoDB

Suivez ces instructions pour télécharger et installer NoSQL Workbench et DynamoDB local* pour Amazon DynamoDB.

Prérequis

Deux logiciels sont requis pour les installations d'Ubuntu : libfuse2 et curl.

libfuse2

Depuis Ubuntu 22.04, libfuse2 n'est plus installé par défaut. Pour résoudre ce problème, exécutez `sudo add-apt-repository universe && sudo apt install libfuse2` pour l'installer pour la dernière version d'Ubuntu.

curl

Mettez à jour Ubuntu, exécutez `sudo apt update && sudo apt upgrade`

Ensuite, installez cURL, exécutez : `sudo apt install curl`

Pour télécharger NoSQL Workbench et DynamoDB local

1. Téléchargez la version appropriée de NoSQL Workbench pour votre système d'exploitation.

Système d'exploitation	Lien de téléchargement
macOS (Intel)**	Téléchargement pour macOS (Intel)
macOS (silicium Apple)	Téléchargement pour macOS (silicium Apple)
Windows	Téléchargement pour Windows
Linux***	Téléchargement pour Linux

* NoSQL Workbench inclut DynamoDB local en tant que composant facultatif du processus d'installation.

** Si un message d'avertissement apparaît lorsque vous essayez d'ouvrir NoSQL Workbench indiquant que l'application n'est pas inscrite auprès d'Apple par un développeur identifié, procédez comme suit :

1. Localisez l'application, puis ouvrez-la.
2. Cliquez sur l'icône de l'application en maintenant la touche Ctrl enfoncée, puis choisissez Ouvrir dans le menu contextuel.

   Cela enregistre l'application en tant qu'exception aux paramètres de sécurité. Ouvrez l'application en double-cliquant dessus, comme vous ouvriez n'importe quelle application inscrite.

*** NoSQL Workbench prend en charge Ubuntu 12.04, Fedora 21 et Debian 8, ou toute version plus récente de ces distributions Linux.

2. Lancez l'application téléchargée et suivez les instructions pour installer NoSQL Workbench.

> **Note**
>
> La version 11.x ou une version plus récente de l'environnement d'exécution Java (JRE) est nécessaire pour exécuter DynamoDB local.

Installer NoSQL Workbench pour DynamoDB

Suivez ces étapes pour installer NoSQL Workbench et DynamoDB local sur une plateforme compatible.

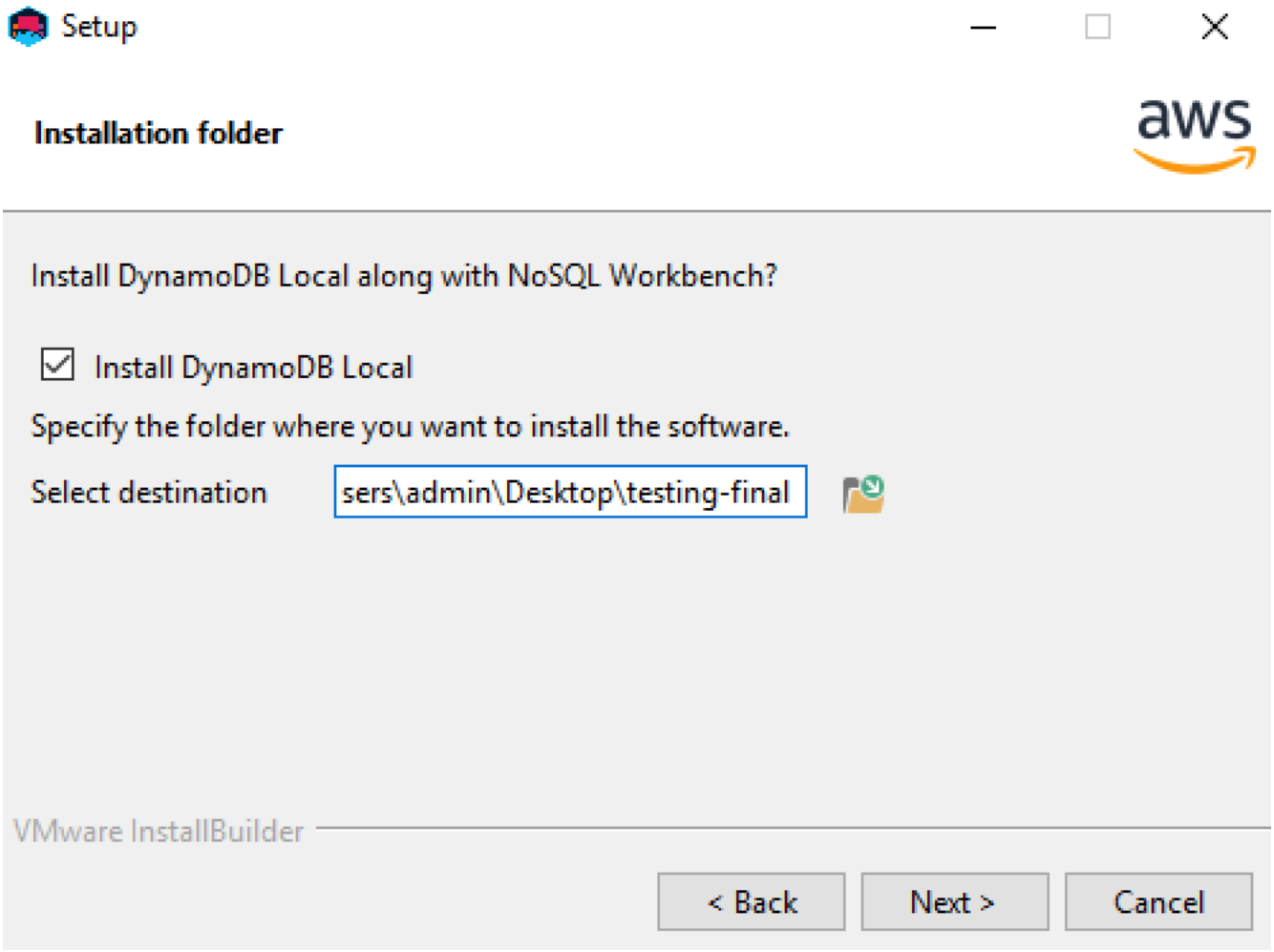
Windows

Pour installer NoSQL Workbench sur Windows

1. Exécutez l'application du programme d'installation de NoSQL Workbench et choisissez la langue de configuration. Choisissez ensuite OK pour commencer la configuration. Pour en savoir plus sur le téléchargement de NoSQL Workbench, consultez Télécharger NoSQL Workbench pour DynamoDB.
2. Choisissez Next (Suivant) pour poursuivre la configuration, puis choisissez à nouveau Next (Suivant) sur l'écran suivant.
3. Par défaut, la case Installer DynamoDB Local est cochée pour inclure DynamoDB local dans le cadre de l'installation. Si vous maintenez cette option sélectionnée, vous vous assurez que DynamoDB local sera installé et que le chemin de destination sera le même que le chemin d'installation de NoSQL Workbench. Si vous décochez la case correspondant à cette option, l'installation de DynamoDB local sera ignorée, et le chemin d'installation ne concernera que NoSQL Workbench.

   Choisissez la destination où vous souhaitez installer le logiciel, puis cliquez sur Next (Suivant).

   > **Note**
   >
   > Si vous avez choisi de ne pas inclure DynamoDB local dans la configuration, décochez la case Installer DynamoDB Local, choisissez Suivant et passez à l'étape 6. Vous pourrez télécharger DynamoDB local séparément en tant qu'installation autonome ultérieurement. Pour plus d'informations, consultez Configuration de DynamoDB Local (version téléchargeable) .



4. Choisissez le numéro de port que DynamoDB local doit utiliser. La valeur par défaut du port est 8000. Après avoir saisi le numéro de port, choisissez Next (Suivant).
5. Choisissez Next (Suivant) pour commencer la configuration.
6. Lorsque la configuration est terminée, choisissez Finish (Terminer) pour fermer l'écran de configuration.
7. Ouvrez l'application dans votre chemin d'installation, par exemple /programs/DynamoDBWorkbench/.

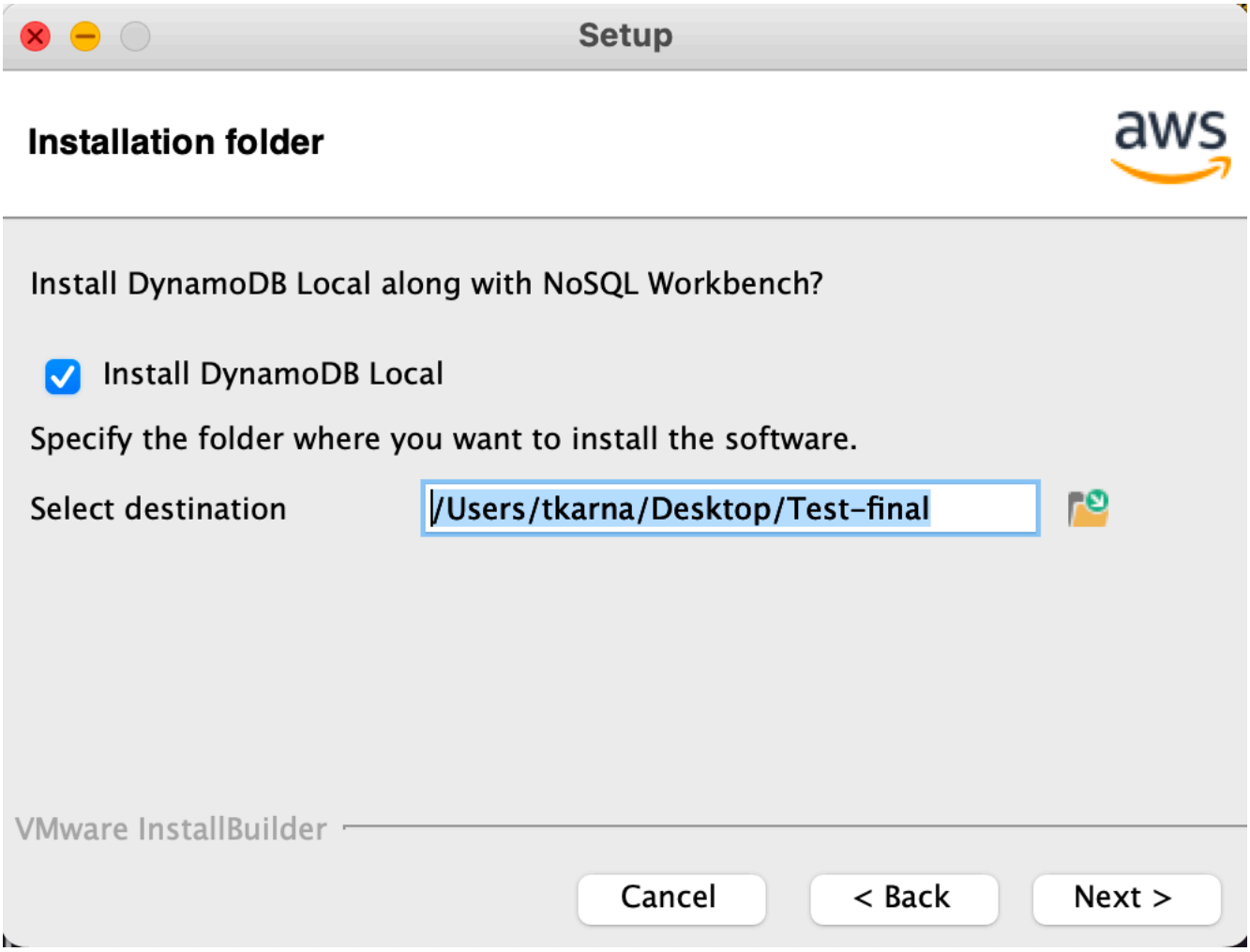
macOS

Pour installer NoSQL Workbench sur macOS

1. Exécutez l'application du programme d'installation de NoSQL Workbench et choisissez la langue de configuration. Choisissez ensuite OK pour commencer la configuration. Pour en savoir plus sur le téléchargement de NoSQL Workbench, consultez Télécharger NoSQL Workbench pour DynamoDB.
2. Choisissez Next (Suivant) pour poursuivre la configuration, puis choisissez à nouveau Next (Suivant) sur l'écran suivant.
3. Par défaut, la case Installer DynamoDB local est cochée pour inclure DynamoDB local dans le cadre de l'installation. Si vous maintenez cette option sélectionnée, vous vous assurez que DynamoDB local sera installé et que le chemin de destination sera le même que le chemin d'installation de NoSQL Workbench. Si vous décochez cette case, l'installation de DynamoDB local sera ignorée, et le chemin d'installation ne concernera que NoSQL Workbench.

   Choisissez la destination où vous souhaitez installer le logiciel, puis cliquez sur Next (Suivant).

   > **Note**
   >
   > Si vous avez choisi de ne pas inclure DynamoDB local dans la configuration, décochez la case Installer DynamoDB Local, choisissez Suivant et passez à l'étape 6. Vous pourrez télécharger DynamoDB local séparément en tant qu'installation autonome ultérieurement. Pour plus d'informations, consultez Configuration de DynamoDB Local (version téléchargeable) .



4. Choisissez le numéro de port que DynamoDB local doit utiliser. La valeur par défaut du port est 8000. Après avoir saisi le numéro de port, choisissez Next (Suivant).
5. Choisissez Next (Suivant) pour commencer la configuration.
6. Lorsque la configuration est terminée, choisissez Finish (Terminer) pour fermer l'écran de configuration.
7. Ouvrez l'application dans votre chemin d'installation, par exemple /Applications/DynamoDBWorkbench/.

> **Note**
>
> NoSQL Workbench pour macOS effectue des mises à jour automatiques. Pour recevoir des notifications concernant les mises à jour, activez l'accès de NoSQL Workbench

aux notifications dans System Preferences > Notifications (Préférences système > Notifications).

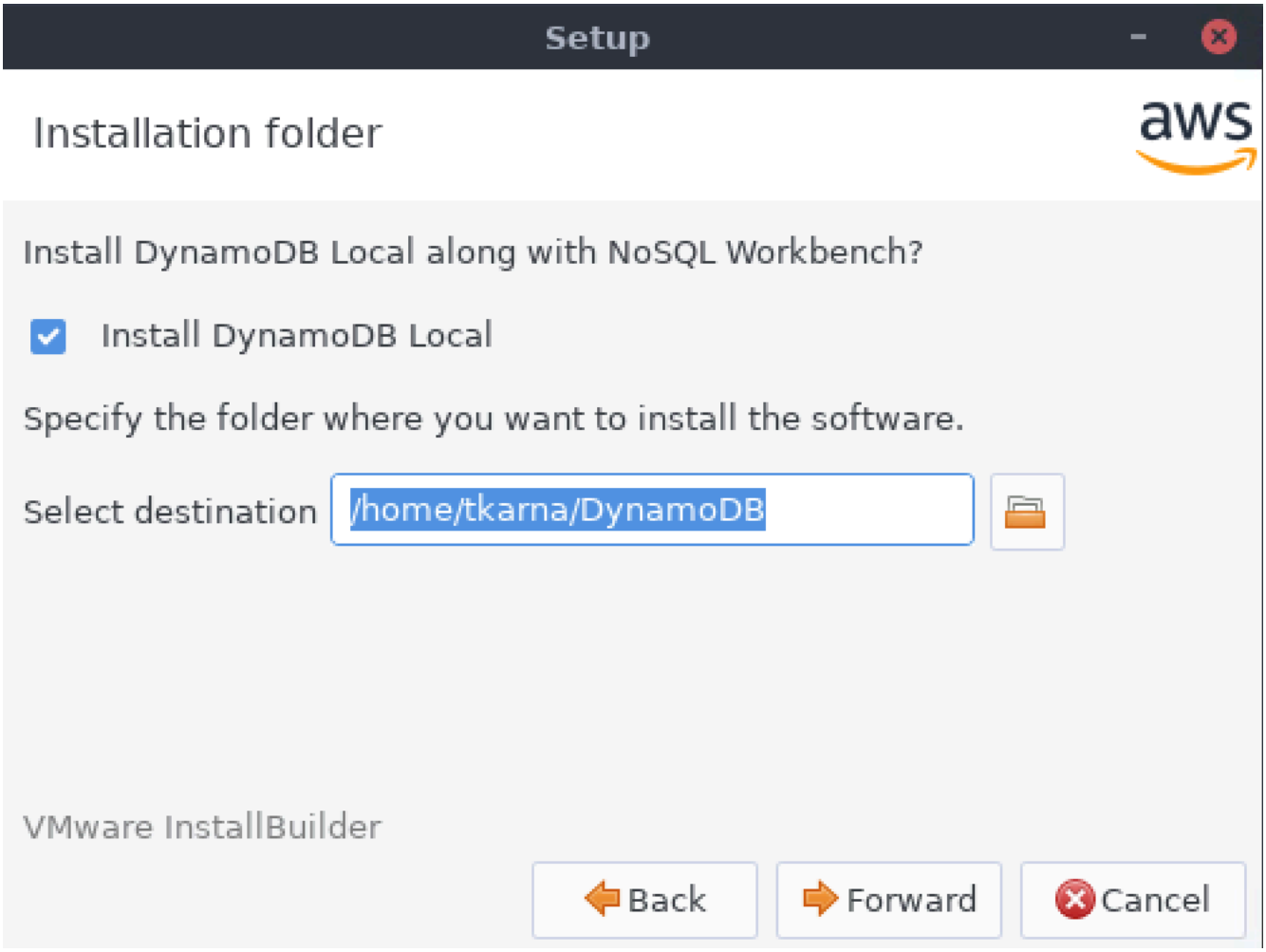
Linux

Pour installer NoSQL Workbench sur Linux

1. Exécutez l'application du programme d'installation de NoSQL Workbench et choisissez la langue de configuration. Choisissez ensuite OK pour commencer la configuration. Pour en savoir plus sur le téléchargement de NoSQL Workbench, consultez Télécharger NoSQL Workbench pour DynamoDB.
2. Choisissez Forward (Suivant) pour poursuivre la configuration, puis Forward (Suivant) sur l'écran suivant.
3. Par défaut, la case Installer DynamoDB local est cochée pour inclure DynamoDB local dans le cadre de l'installation. Si vous maintenez cette option sélectionnée, vous vous assurez que DynamoDB local sera installé et que le chemin de destination sera le même que le chemin d'installation de NoSQL Workbench. Si vous décochez cette case, l'installation de DynamoDB local sera ignorée, et le chemin d'installation ne concernera que NoSQL Workbench.

   Choisissez la destination où vous souhaitez installer le logiciel, puis choisissez Forward (Suivant).

   > **Note**
   >
   > Si vous avez choisi de ne pas inclure DynamoDB local dans la configuration, décochez la case Installer DynamoDB local, choisissez Suivant et passez à l'étape 6. Vous pourrez télécharger DynamoDB local séparément en tant qu'installation autonome ultérieurement. Pour plus d'informations, consultez Configuration de DynamoDB Local (version téléchargeable) .



4. Choisissez le numéro de port que DynamoDB local doit utiliser. La valeur par défaut du port est 8000. Après avoir saisi le numéro de port, choisissez Forward (Suivant).
5. Choisissez Forward (Suivant) pour commencer la configuration.
6. Lorsque la configuration est terminée, choisissez Finish (Terminer) pour fermer l'écran de configuration.
7. Ouvrez l'application dans votre chemin d'installation, par exemple /usr/local/programs/DynamoDBWorkbench/.

Pour démarrer une AppImage sous Linux

1. Rendez le fichier AppImage exécutable :

   chmod +x noSQL-workbench-linux.AppImage

Remplacez noSQL-workbench-linux.AppImage par le nom de fichier réel de l'AppImage que vous avez téléchargée.

2. Exécutez l'AppImage :

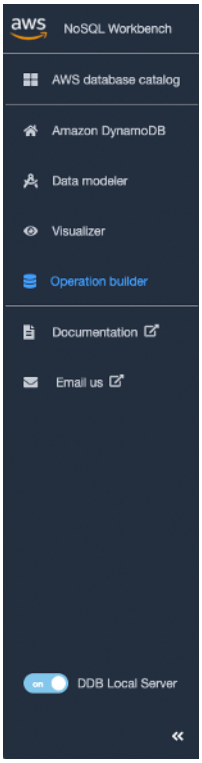
   ./noSQL-workbench-linux.AppImage

   Cela lance l'application NoSQL Workbench.

> **Note**
>
> En fonction de votre distribution Linux, vous devrez peut-être installer des dépendances supplémentaires pour que l'AppImage fonctionne correctement. Si vous rencontrez des problèmes, consultez la documentation fournie par les développeurs d'AppImage ou demandez de l'aide à la communauté.

> **Note**
>
> Si vous avez choisi d'installer DynamoDB local dans le cadre de l'installation de NoSQL Workbench, DynamoDB local sera préconfiguré avec les options par défaut. Pour changer les options par défaut, modifiez le script DDBlocalStart situé dans le répertoire /resources/DDBlocal_Scripts/. Vous le trouverez sous le chemin que vous avez indiqué lors de l'installation. Pour en savoir plus sur les options de DynamoDB local, consultez Notes d'utilisation de DynamoDB Local .

Si vous avez choisi d'installer DynamoDB local dans le cadre de l'installation de NoSQL Workbench, un bouton est à votre disposition pour activer ou désactiver DynamoDB local, comme indiqué dans l'image suivante.



Création de modèles de données avec NoSQL Workbench

Vous pouvez utiliser le modélisateur de données dans NoSQL Workbench pour Amazon DynamoDB afin de générer des modèles de données ou de concevoir des modèles à partir de modèles existants, qui pourront satisfaire aux modèles d'accès aux données de votre application. Le modélisateur de données comprend quelques exemples de modèles de données pour vous aider à démarrer.

Rubriques

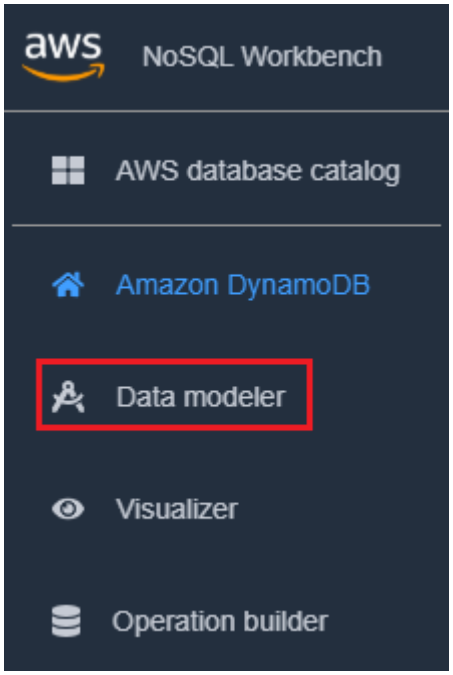
- Création d'un modèle de données
- Importation d'un modèle de données existant
- Exportation d'un modèle de données
- Modification d'un modèle de données existant

Création d'un modèle de données

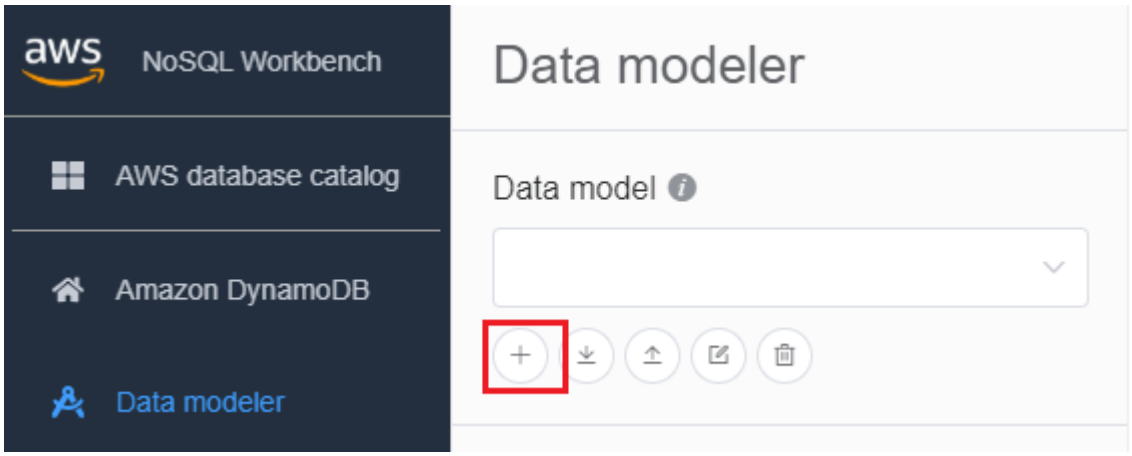
Procédez comme suit pour créer un modèle de données dans Amazon DynamoDB à l'aide de NoSQL Workbench.

Pour créer un modèle de données

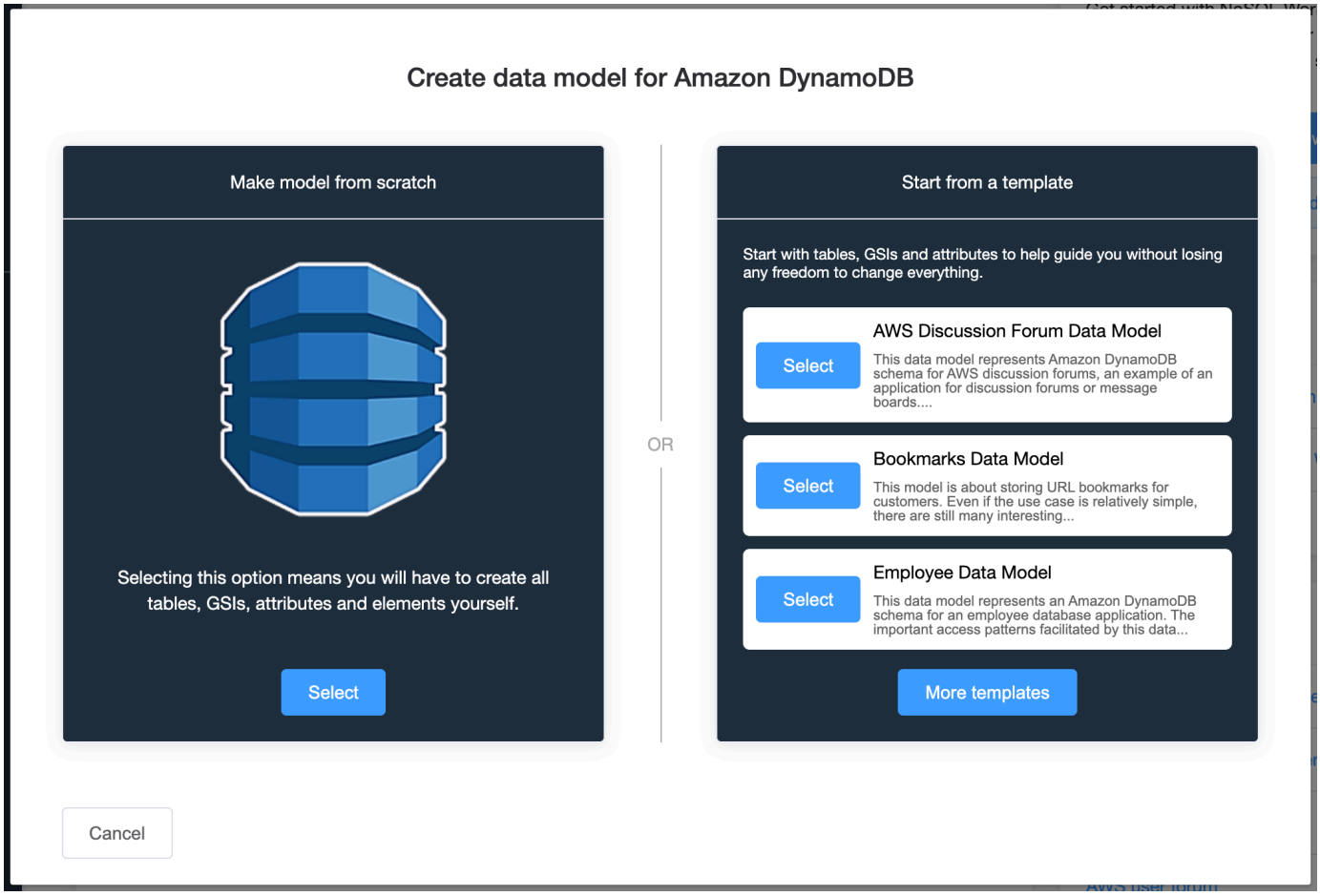
1. Ouvrez NoSQL Workbench et dans le panneau de navigation sur la gauche, choisissez le bouton Data modeler (Modélisateur de données).



2. Choisissez Create data model (Créer un modèle de données).

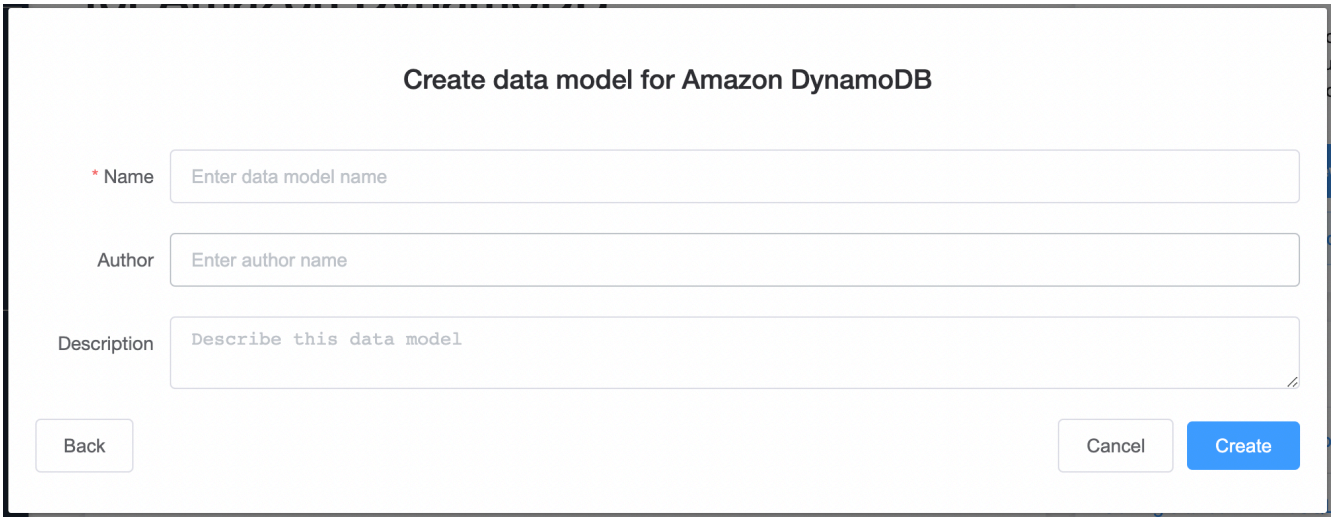


L'option Create data model (Créer un modèle de données) propose deux options : créer un modèle à partir de zéro et créer un modèle à partir d'un modèle.



Make model from scratch

Pour créer un modèle à partir de zéro, entrez un nom, un auteur et une description pour le modèle de données. Lorsque vous avez terminé, choisissez Create (Créer).



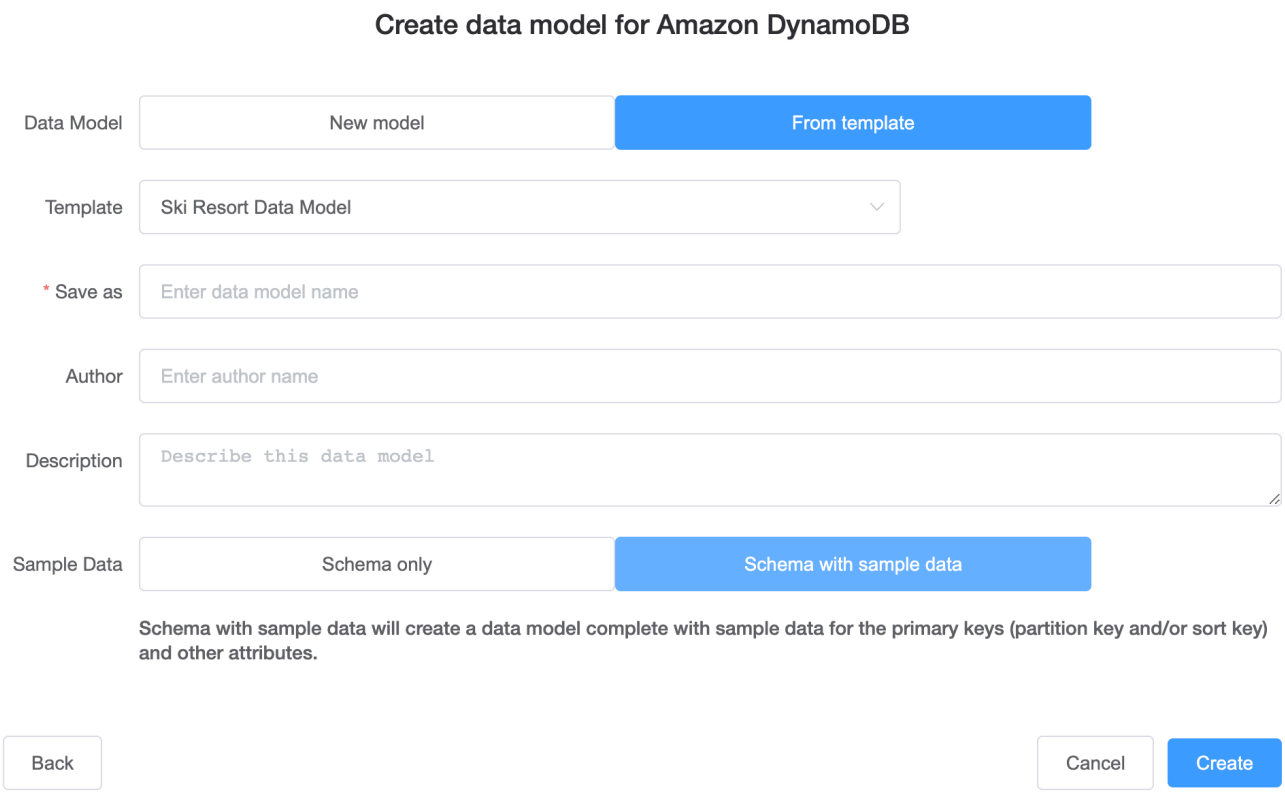
Start from a template

Vous pouvez partir d'un exemple de modèle pour en créer un autre. Choisissez More templates (Plus de modèles) pour voir plus d'options de modèles. Choisissez Select (Sélectionner) pour rechercher le modèle que vous souhaitez utiliser.

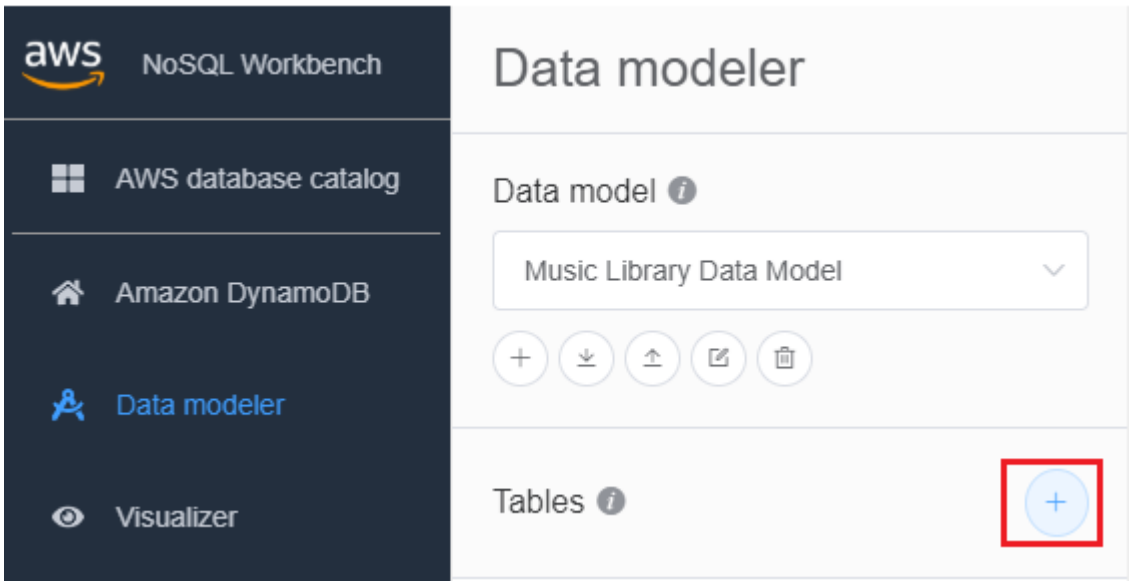
Entrez le nom, l'auteur et la description du modèle de données que vous avez sélectionné. Vous pouvez choisir entre Schema only (Schéma uniquement) et Schema with sample data (Schéma avec des exemples de données).

- L'option Schema only (Schéma uniquement) crée un modèle de données vide avec la clé primaire (clé de partition et clé de tri) et d'autres attributs.
- L'option Schema with sample data (Schéma avec des exemples de données) crée un modèle de données complet avec des exemples de données pour la clé primaire (clé de partition et clé de tri) et d'autres attributs.

Lorsque ces informations sont complètes, choisissez Create (Créer) pour créer le modèle.



3. Une fois le modèle créé, choisissez Add table (Ajouter une table).



Pour plus d'informations, consultez Utilisation des tables dans DynamoDB.

4. Spécifiez les paramètres suivants :

   - Table name (Nom de table) – Saisissez un nom unique pour la table.
   - Clé de partition – Saisissez un nom de clé de partition et spécifiez son type. De même, vous pouvez éventuellement sélectionner un format de type de données plus détaillé pour la génération de données d'exemple.
   - Si vous souhaitez ajouter une clé de tri :

     1. Sélectionnez Add sort key.
     2. Spécifiez le nom de la clé de tri et son type. Vous pouvez éventuellement sélectionner un format de type de données plus détaillé pour la génération de données d'exemple.

> **Note**
>
> Pour en savoir plus sur la conception des clés primaires, la conception et l'utilisation efficaces des clés de partition et l'utilisation des clés de tri, consultez les rubriques suivantes :
>
> - Clé primaire
> - Bonnes pratiques pour la conception et l'utilisation performantes de clés de partition dans DynamoDB

- Bonnes pratiques concernant l'utilisation de clés de tri pour organiser les données dans DynamoDB

5. Pour ajouter d'autres attributs, procédez comme suit pour chacun d'entre eux :

   1. Choisissez Ajouter un attribut.
   2. Spécifiez le nom de l'attribut et son type. Vous pouvez éventuellement sélectionner un format de type de données plus détaillé pour la génération de données d'exemple.

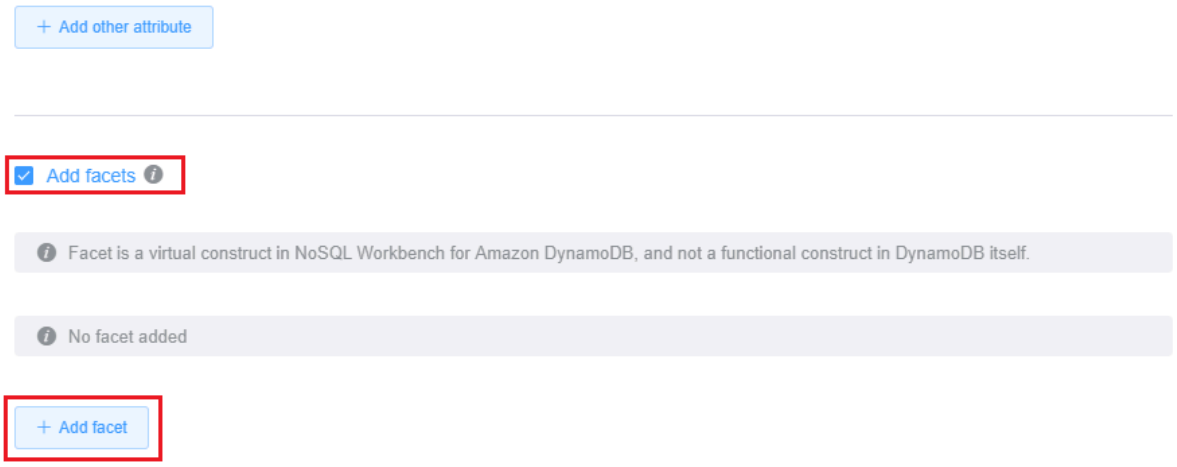
6. Ajouter une facette :

   Si vous le souhaitez, ajoutez une facette. Une facette est une construction virtuelle dans NoSQL Workbench. Il ne s'agit pas d'une construction fonctionnelle dans DynamoDB lui-même.

   > **Note**
   >
   > Dans NoSQL Workbench, les facettes vous permettent de mieux visualiser les différents modèles d'accès aux données d'une application pour Amazon DynamoDB avec seulement un sous-ensemble des données d'une table. Pour en savoir plus sur les facettes, consultez Affichage des modèles d'accès aux données.

   Pour ajouter une facette,

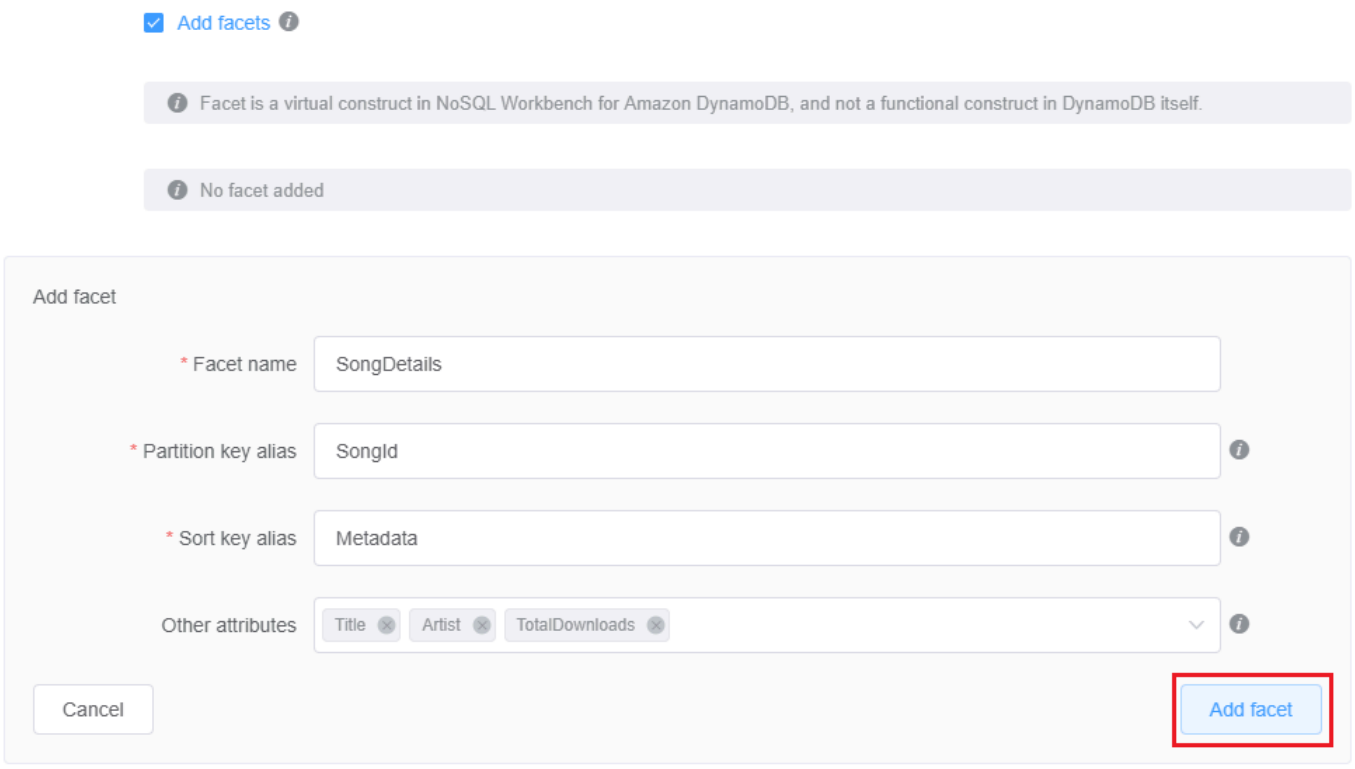
   - Sélectionnez Ajouter des facettes.
   - Choisissez Add Facet (Ajouter une facette).



   - Spécifiez les paramètres suivants :

- Le nom de la facette.
- Un alias de clé de partition pour mieux distinguer la vue de cette facette.
- Un alias de clé de tri.
- Sélectionnez les Autres attributs qui font partie de cette facette.

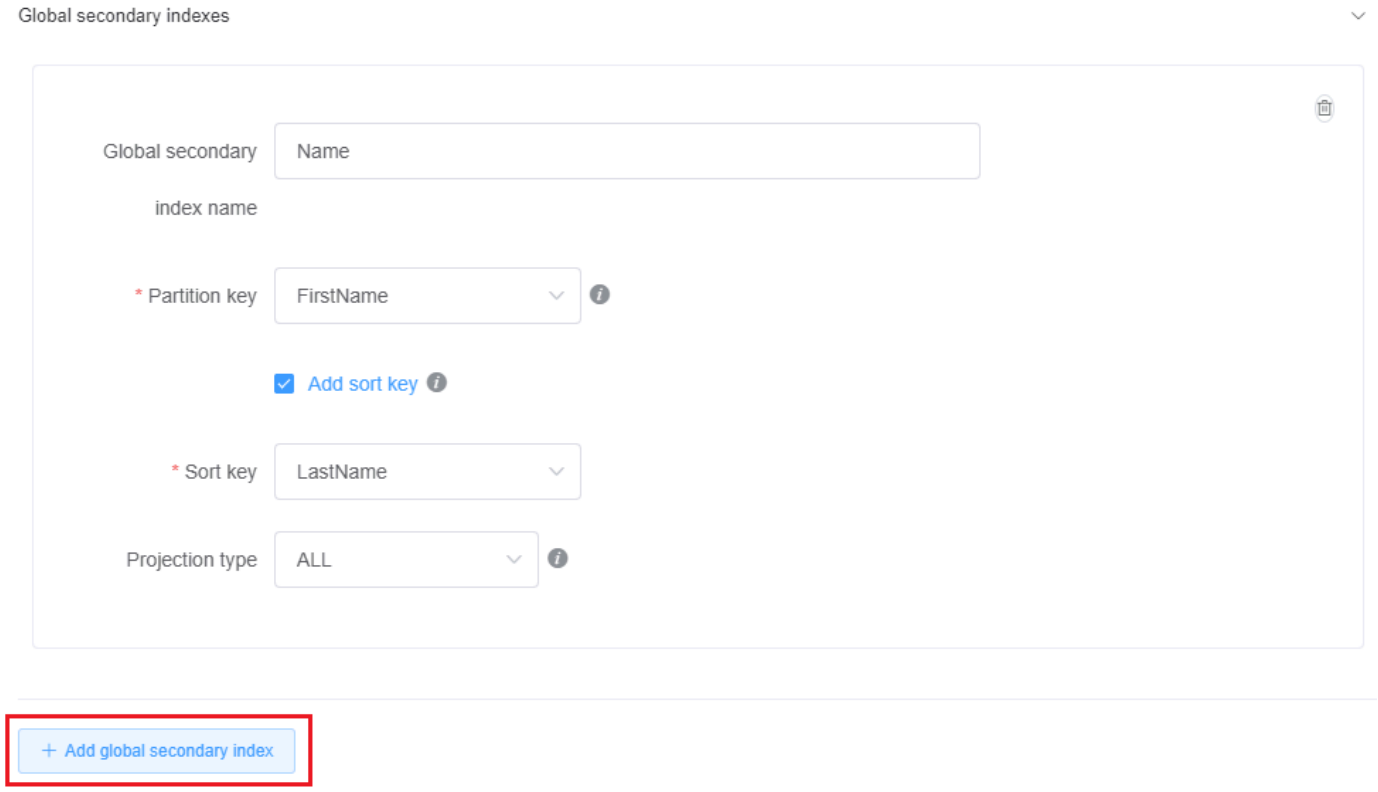
Choisissez Add Facet (Ajouter une facette).



Répétez cette étape si vous souhaitez ajouter d'autres facettes.

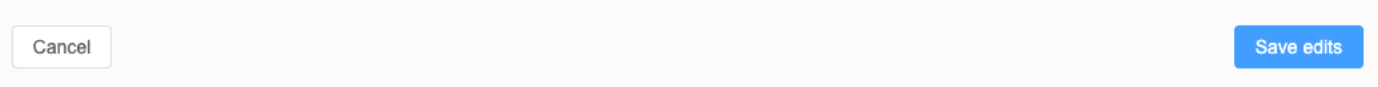
7. Si vous souhaitez ajouter un index secondaire global, choisissez Add global secondary index (Ajouter un index secondaire global).

   Spécifiez le Global secondary index name (Nom de l'index secondaire global), l'attribut Partition key (Clé de partition) et le Projection type (Type de projection).



Pour plus d'informations sur l'utilisation des index secondaires globaux dans DynamoDB, consultez Index secondaires globaux.

8. Enregistrez les modifications apportées aux paramètres de votre table.



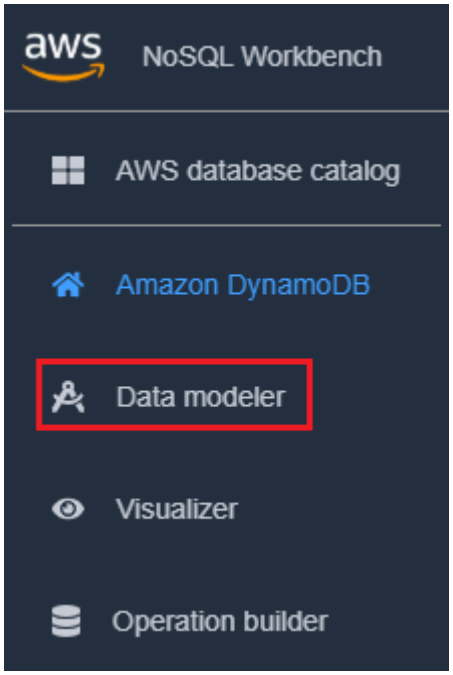
Pour plus d'informations sur l'opération d'API `CreateTable`, consultez CreateTable dans la Référence d'API Amazon DynamoDB.

Importation d'un modèle de données existant

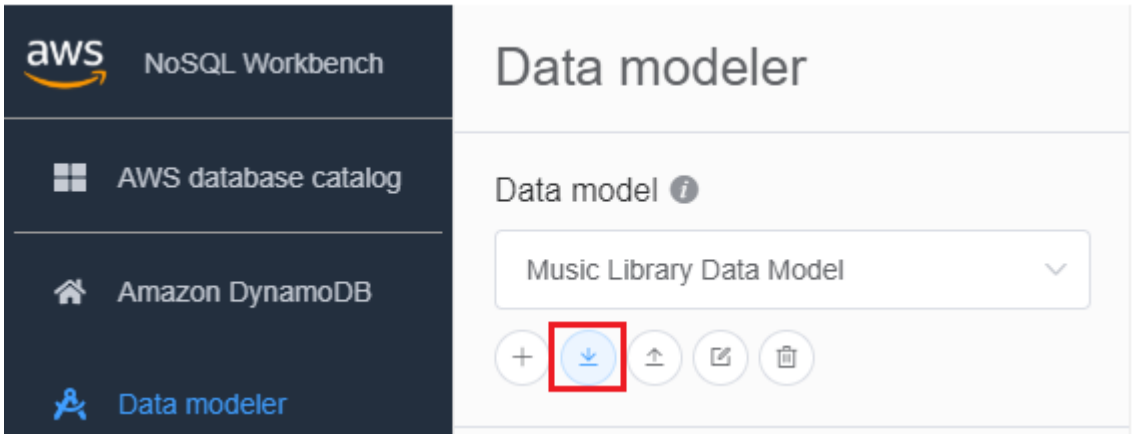
Vous pouvez utiliser NoSQL Workbench pour Amazon DynamoDB afin de générer un modèle de données en important et en modifiant un modèle existant. Vous pouvez importer des modèles de données au format de modèle NoSQL Workbench ou au Format de modèle JSON AWS CloudFormation.

Pour importer un modèle existant

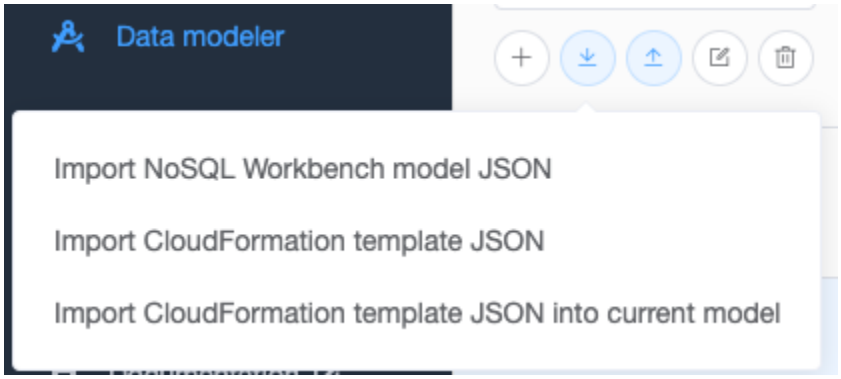
1. Dans NoSQL Workbench et dans le panneau de navigation sur la gauche, choisissez l'icône Data modeler (Modélisateur de données).



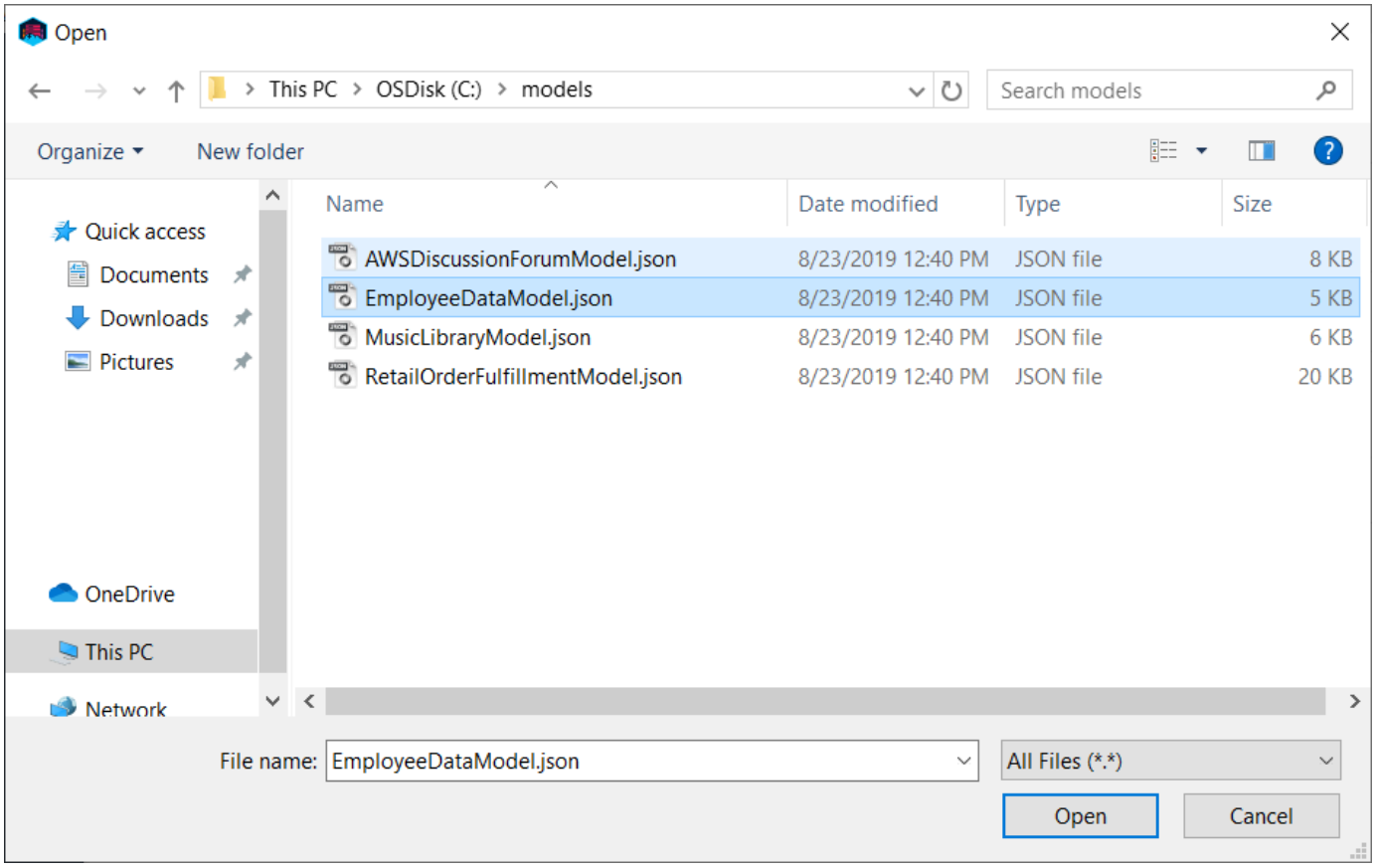
2. Passez le curseur sur Import data model (Importer un modèle de données).



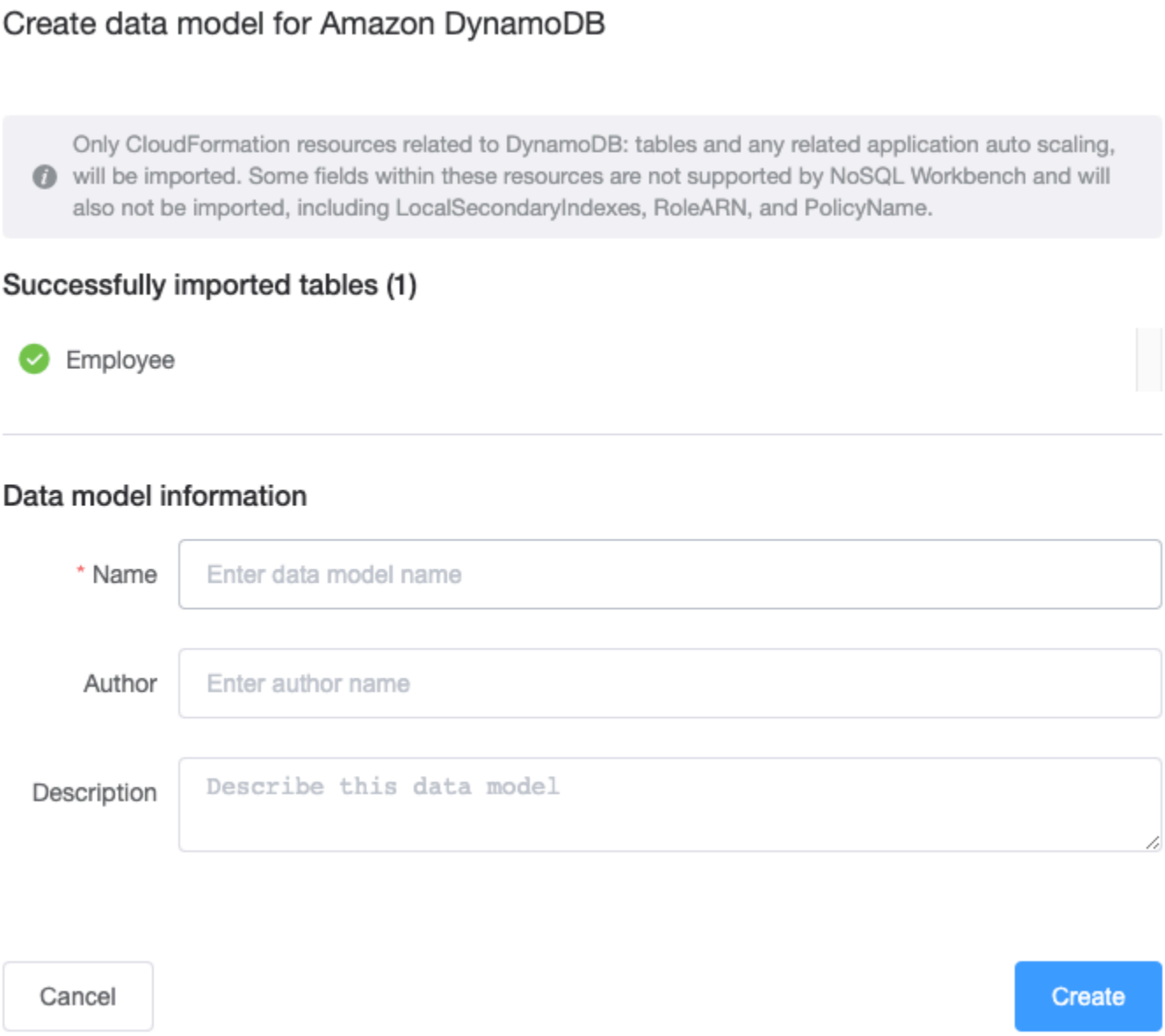
Dans la liste déroulante, choisissez si le modèle à importer est au format de modèle NoSQL Workbench ou au format de modèle JSON CloudFormation. Si un modèle de données existant est ouvert dans NoSQL Workbench, vous avez la possibilité d'importer un modèle CloudFormation dans le modèle actuel.



3. Choisissez un modèle à importer.



4. Si le modèle que vous importez est au format de modèle CloudFormation, vous pouvez voir la liste des tables à importer et avez la possibilité de spécifier un nom, un auteur et une description du modèle de données.

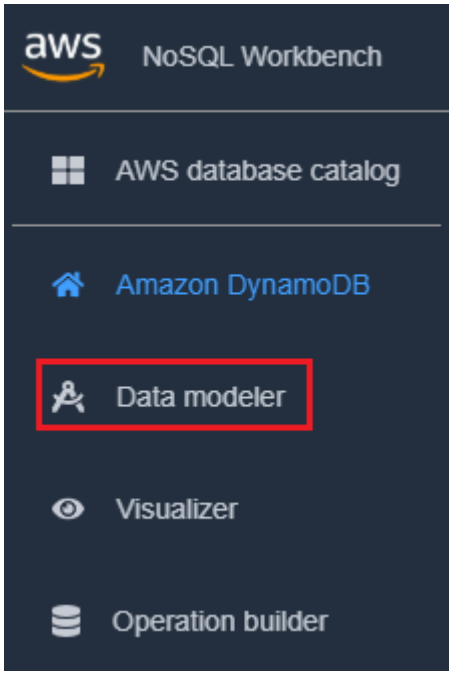


Exportation d'un modèle de données

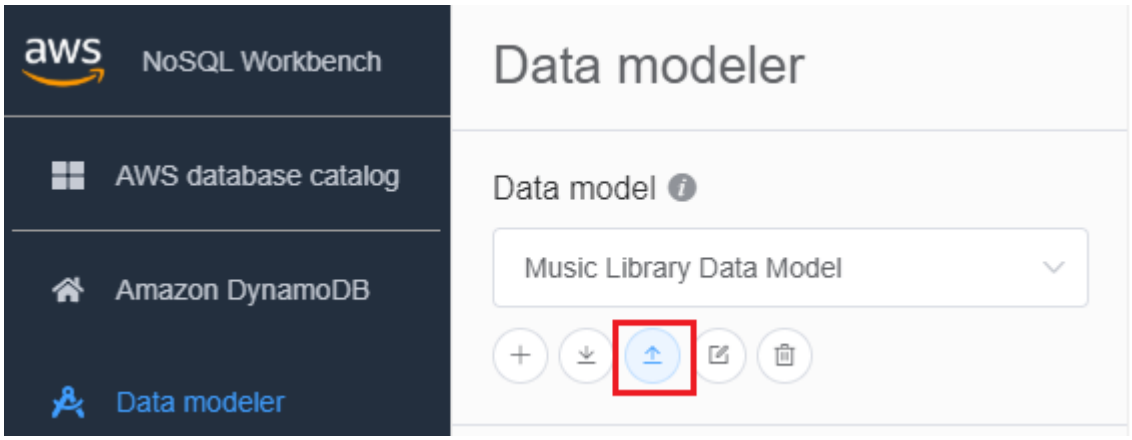
Après avoir créé un modèle de données à l'aide de NoSQL Workbench pour Amazon DynamoDB, vous pouvez enregistrer et exporter le modèle au format de modèle NoSQL Workbench ou au Format de fichier JSON AWS CloudFormation.

Pour exporter un modèle de données

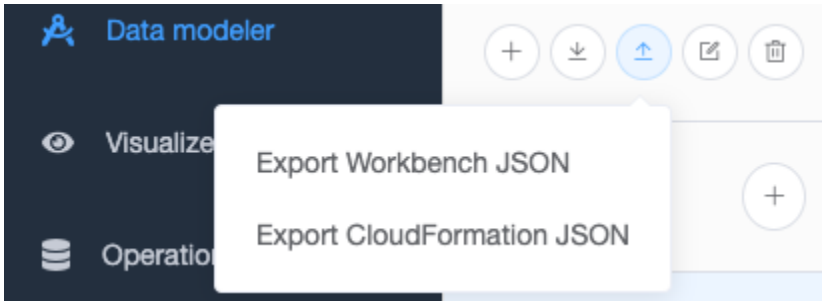
1. Dans NoSQL Workbench et dans le panneau de navigation sur la gauche, choisissez l'icône Data modeler (Modélisateur de données).



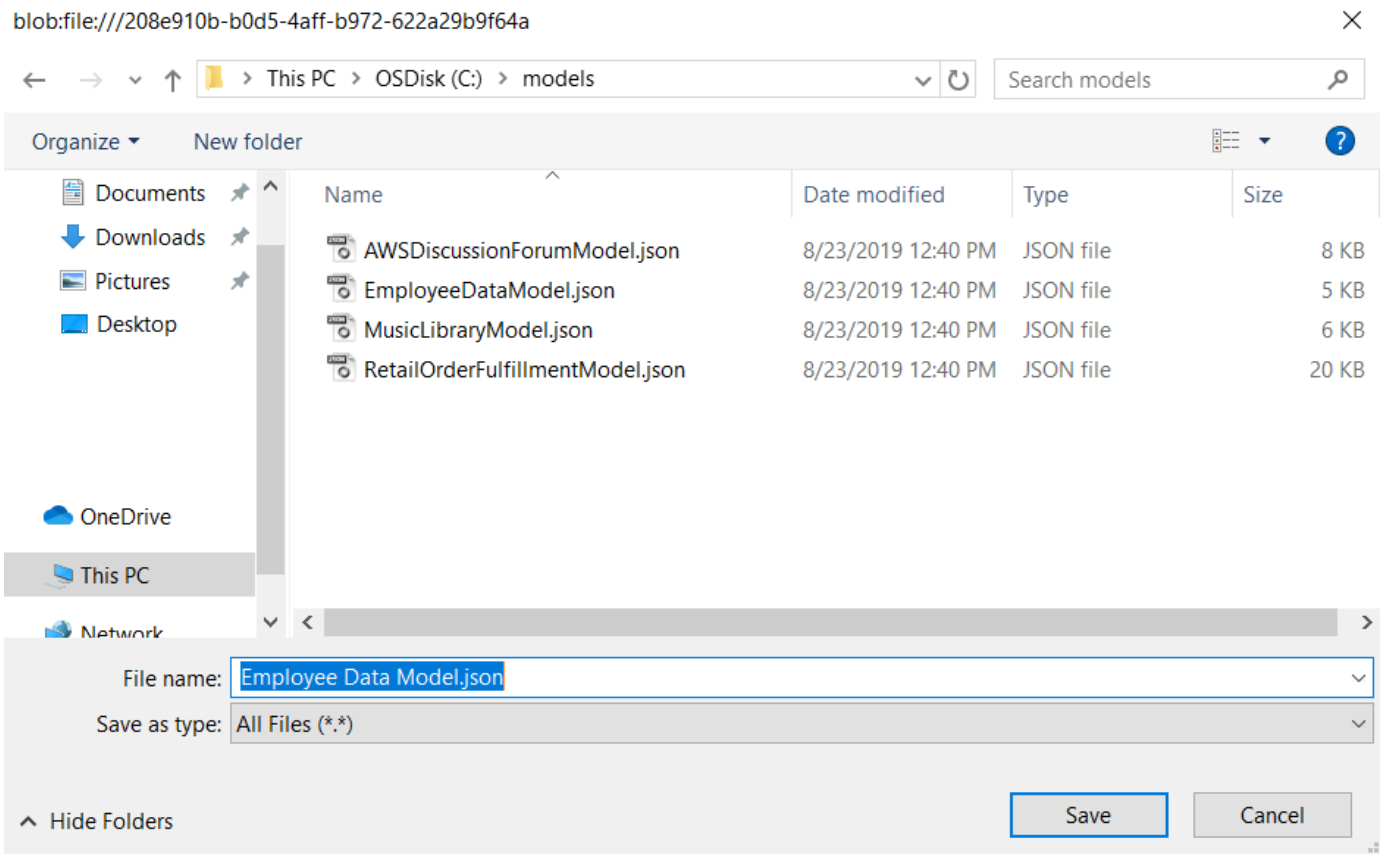
2. Passez le curseur sur Export data model (Exporter un modèle de données).



Dans la liste déroulante, choisissez d'exporter votre modèle de données au format de modèle NoSQL Workbench ou au format de modèle JSON CloudFormation.



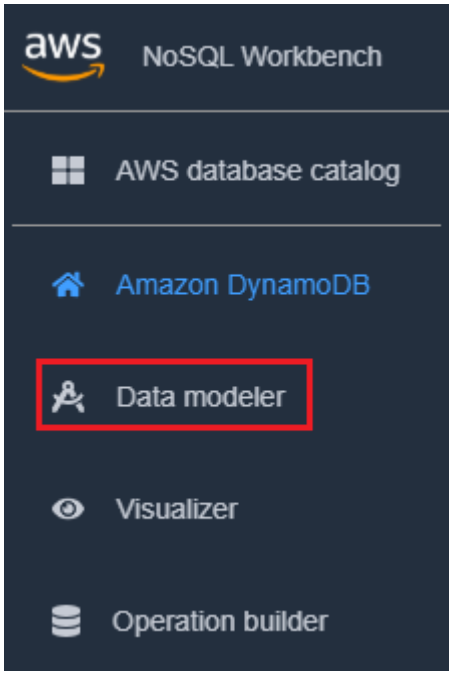
3. Choisissez un emplacement pour enregistrer votre modèle.



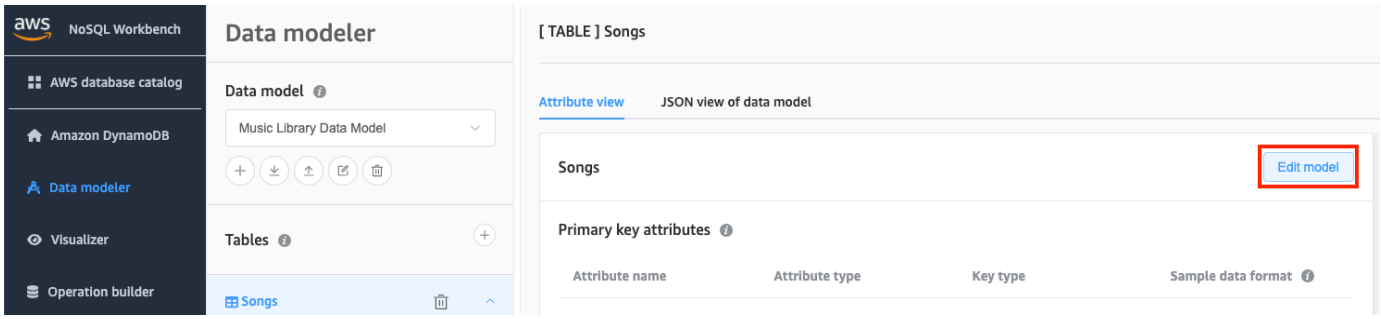
Modification d'un modèle de données existant

Pour modifier un modèle existant

1. Dans NoSQL Workbench, dans le volet de navigation sur la gauche, choisissez le bouton Data modeler (modélisateur de données).



2. Sélectionnez le modèle de données et choisissez la table que vous souhaitez modifier. Choisissez Modifier le modèle.



3. Apportez les modifications nécessaires, puis choisissez Save edits (Enregistrer les modifications).

Pour modifier manuellement un modèle existant et ajouter une facette

1. Exportez votre modèle Pour plus d'informations, consultez Exportation d'un modèle de données.
2. Ouvrez le fichier exporté dans une éditeur.
3. Localisez l'objet `DataModel` de la table pour laquelle vous souhaitez créer une facette.

   Ajoutez un tableau `TableFacets` représentant toutes les facettes de la table.

   Pour chaque facette, ajoutez un objet à la table `TableFacets`. Chaque élément du tableau possède les propriétés suivantes :

- `FacetName` – Nom pour votre facette. Cette valeur doit être unique dans le modèle.
- `PartitionKeyAlias` – Nom convivial de la clé de partition de la table. Cet alias s'affiche lorsque vous voyez la facette dans NoSQL Workbench.
- `SortKeyAlias` – Nom convivial de la clé de tri de la table. Cet alias s'affiche lorsque vous voyez la facette dans NoSQL Workbench. Cette propriété n'est pas nécessaire si la table n'a pas de clé de tri définie.
- `NonKeyAttributes` – Tableau de noms d'attributs nécessaires pour le modèle d'accès. Ces noms doivent correspondre aux noms d'attribut définis pour la table.

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
```

```
        },
        {
          "AttributeName": "TotalDownloads",
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. Vous pouvez désormais importer le modèle modifié dans NoSQL Workbench. Pour plus d'informations, consultez Importation d'un modèle de données existant.

Visualisation des modèles d'accès de données

Vous pouvez utiliser l'outil de visualisation dans NoSQL Workbench pour Amazon DynamoDB afin de mapper des requêtes et de visualiser les différents modèles d'accès (appelés facettes) d'une application. Chaque facette correspond à un modèle d'accès différent dans DynamoDB. Vous pouvez également ajouter manuellement des données à votre modèle de données ou importer des données à partir de MySQL.

Rubriques

- Ajout d'un exemple de données à un modèle de données
- Importation des exemples de données à partir d'un fichier CSV
- Affichage des modèles d'accès aux données
- Affichage de toutes les tables dans un modèle de données à l'aide de la vue d'agrégation
- Validation d'un modèle de données dans DynamoDB

Ajout d'un exemple de données à un modèle de données

L'ajout d'un exemple de données à votre modèle vous permet d'afficher les données lorsque vous visualisez le modèle et ses différents modèles d'accès de données, ou facettes.

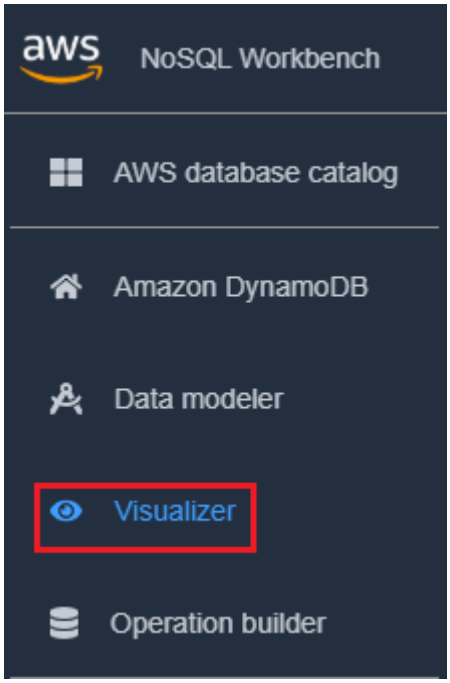
Il existe deux façons d'ajouter des données d'exemple. L'une consiste à utiliser notre outil de génération automatique de données d'exemple. L'autre consiste à ajouter les données en plusieurs fois.

Procédez comme suit pour ajouter un exemple de données à un modèle de données à l'aide de NoSQL Workbench pour Amazon DynamoDB.

Pour générer automatiquement des données d'exemple

La génération automatique de données d'exemple vous permet de générer entre 1 et 5 000 lignes de données pour une utilisation immédiate. Vous pouvez spécifier un type de données d'exemple détaillé pour créer des données réalistes en fonction de vos besoins de conception et de test. Pour pouvoir générer des données réalistes, vous devez spécifier le format du type de données d'exemple pour vos attributs dans le modélisateur de données. Consultez Création d'un modèle de données pour spécifier des formats de type de données d'exemple.

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).

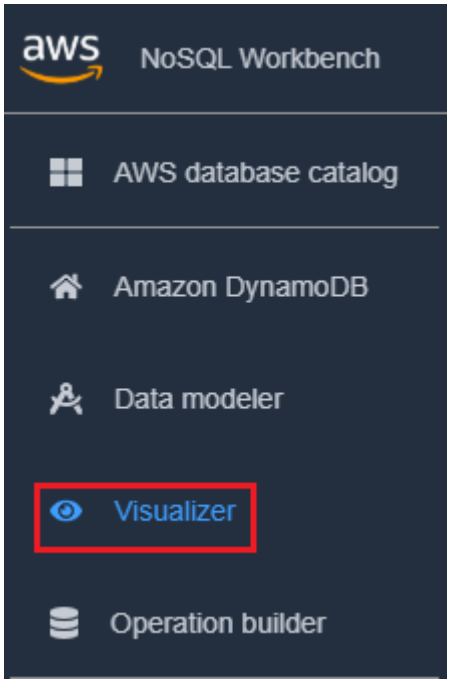


2. Dans le visualiseur, sélectionnez le modèle de données et choisissez la table.
3. Choisissez le menu déroulant Action, puis sélectionnez Ajouter des données d'exemple.

4. Saisissez le nombre ou les éléments de données d'exemple que vous souhaitez générer, puis sélectionnez Confirmer.

Pour ajouter les données d'exemple en plusieurs fois

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).

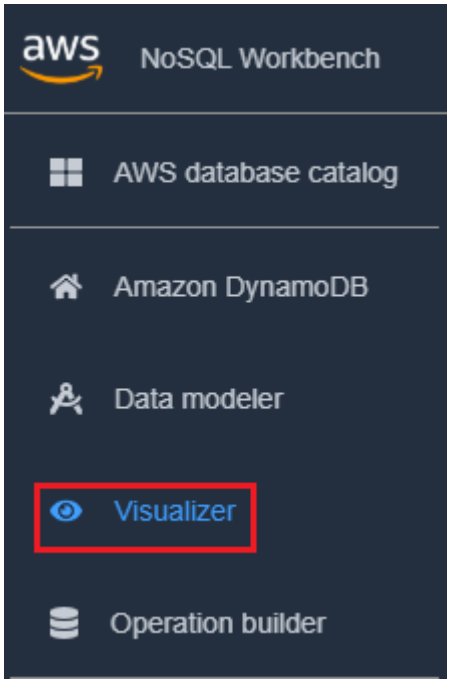


2. Dans le visualiseur, sélectionnez le modèle de données et choisissez la table.
3. Choisissez le menu déroulant Action, puis sélectionnez Modifier les données.

4. Choisissez Ajouter une nouvelle ligne. Saisissez les données d'exemple dans les zones de texte vides, puis choisissez Ajouter une nouvelle ligne pour ajouter des lignes supplémentaires. Lorsque vous avez terminé, choisissez Enregistrer les modifications.

Pour supprimer des données d'exemple

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).



2. Dans le visualiseur, sélectionnez le modèle de données et choisissez la table.
3. Choisissez le menu déroulant Action, puis sélectionnez Modifier les données.

4. Sélectionnez l'icône de suppression en regard de chaque ligne de données que vous souhaitez supprimer.

Importation des exemples de données à partir d'un fichier CSV

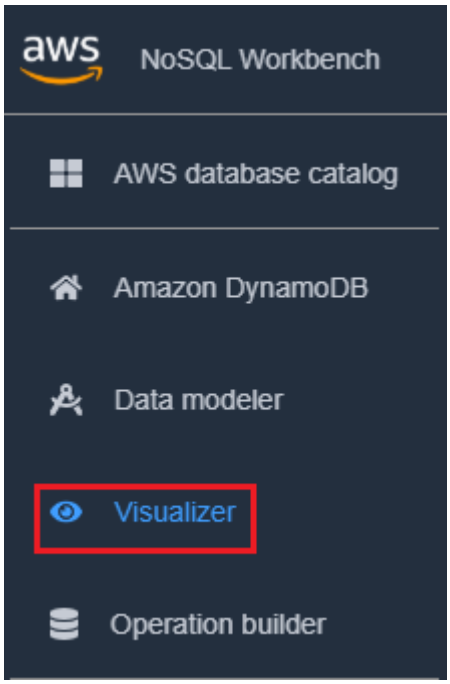
Si vous avez des exemples de données préexistants dans un fichier CSV, vous pouvez les importer dans NoSQL Workbench. Cela vous permet de remplir rapidement votre modèle avec des exemples de données sans avoir à les saisir ligne par ligne.

Les noms de colonnes du fichier CSV doivent correspondre aux noms d'attributs de votre modèle de données. Ils ne doivent pas nécessairement être dans le même ordre. Par exemple, si votre modèle de données comporte des attributs appelés `LoginAlias`, `FirstName` et `LastName`, vos colonnes CSV peuvent être `LastName`, `FirstName` et `LoginAlias`.

L'importation de données à partir d'un fichier CSV est limitée à 150 lignes à la fois.

Pour importer des données à partir d'un fichier CSV dans NoSQL Workbench

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).



2. Dans le visualiseur, sélectionnez le modèle de données et choisissez la table.
3. Choisissez le menu déroulant Action, puis sélectionnez Modifier les données.
4. Choisissez à nouveau le menu déroulant Action, puis sélectionnez Importer un fichier CSV.
5. Sélectionnez le fichier CVS, puis choisissez Ouvrir. Les données du fichier CSV seront ajoutées à votre table.

> **Note**
>
> Si votre fichier CSV contient une ou plusieurs lignes possédant les mêmes clés que les éléments déjà présents dans votre table, vous aurez la possibilité de remplacer les éléments existants ou de les ajouter à la fin du tableau. Si vous choisissez d'ajouter les éléments, le suffixe « -Copy » sera ajouté à la clé de chaque élément dupliqué pour différencier les éléments en double des éléments qui se trouvaient déjà dans le tableau.

Affichage des modèles d'accès aux données

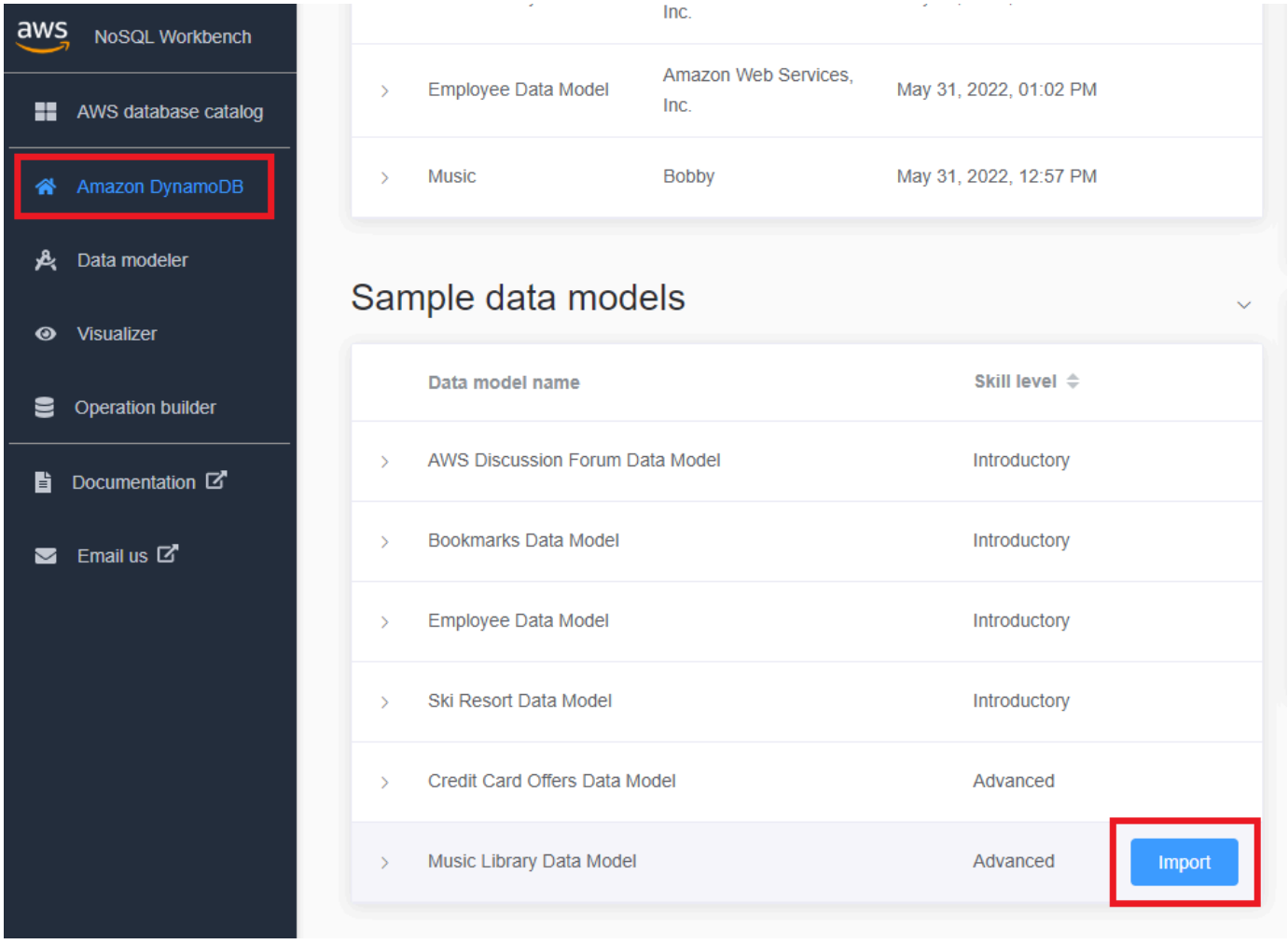
Dans NoSQL Workbench, les facettes représentent les différents modèles d'accès aux données d'une application pour Amazon DynamoDB. Les facettes peuvent vous aider à visualiser votre

modèle de données lorsque plusieurs types de données sont représentés par une clé de tri. Les facettes vous permettent d'afficher un sous-ensemble de données dans une table, sans avoir à voir les enregistrements qui ne répondent pas aux contraintes de la facette. Les facettes sont considérées comme un outil de modélisation de données visuelles et n'existent pas en tant que construction utilisable dans DynamoDB, car elles sont purement une aide à la modélisation des modèles d'accès.

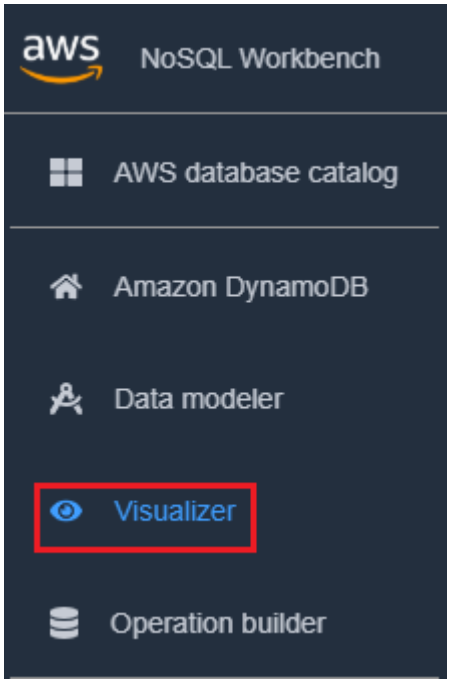
Pour voir un exemple de facettes, vous pouvez importer l'un de nos modèles de données d'exemple constitué de facettes.

Importer un exemples de modèle de données

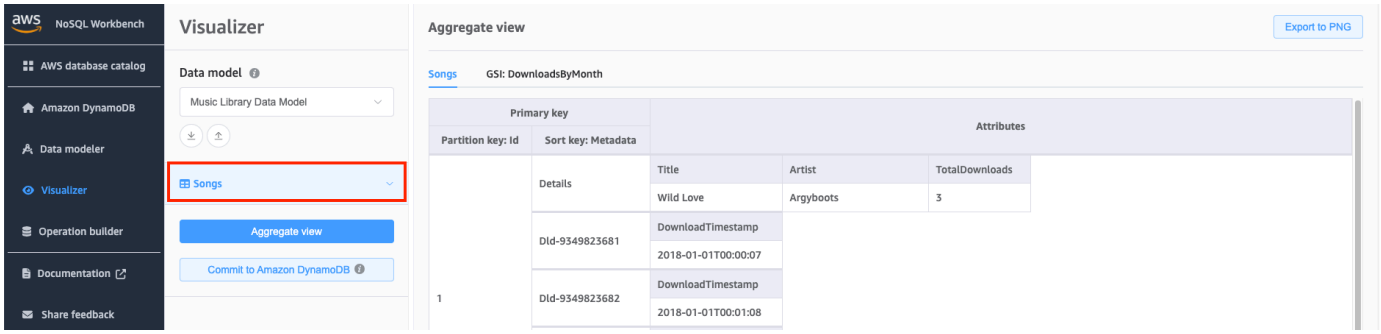
1. Sur la gauche, choisissez Amazon DynamoDB.
2. Dans la section « Sample data models » (Exemples de modèles de données), placez le curseur sur le modèle de données de bibliothèque musicale et choisissez Import (Importer).



3. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).

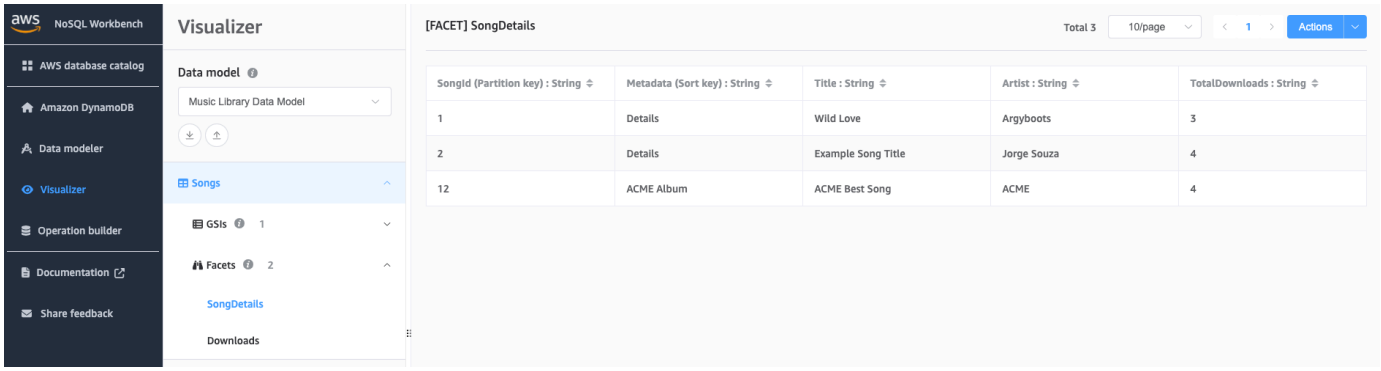


4. Choisissez la table Songs pour l'agrandir Une vue agrégée de vos données s'affiche.



5. Choisissez la flèche déroulante Facets (Facettes) pour développer les facettes disponibles.

6. Choisissez la facette SongDetails pour visualiser les données avec la facette SongDetails appliquée.



Vous pouvez également modifier les définitions de facettes à l'aide du Data Modeler (Modeleur de données). Pour plus d'informations, consultez Modification d'un modèle de données existant.

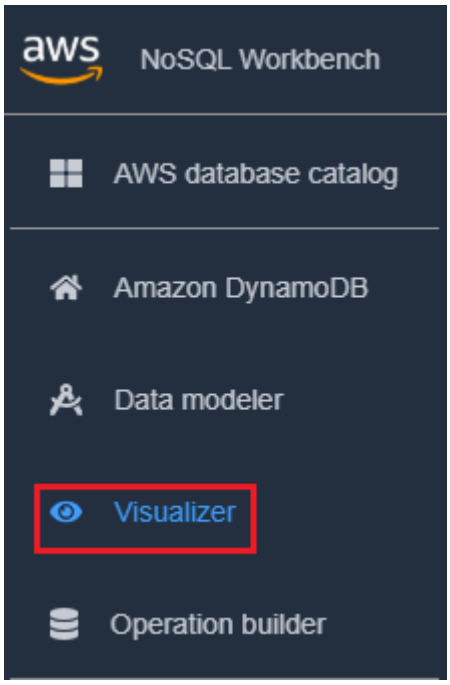
Affichage de toutes les tables dans un modèle de données à l'aide de la vue d'agrégation

La vue d'agrégation dans NoSQL Workbench pour Amazon DynamoDB représente toutes les tables d'un modèle de données. Pour chaque table, les informations suivantes s'affichent :

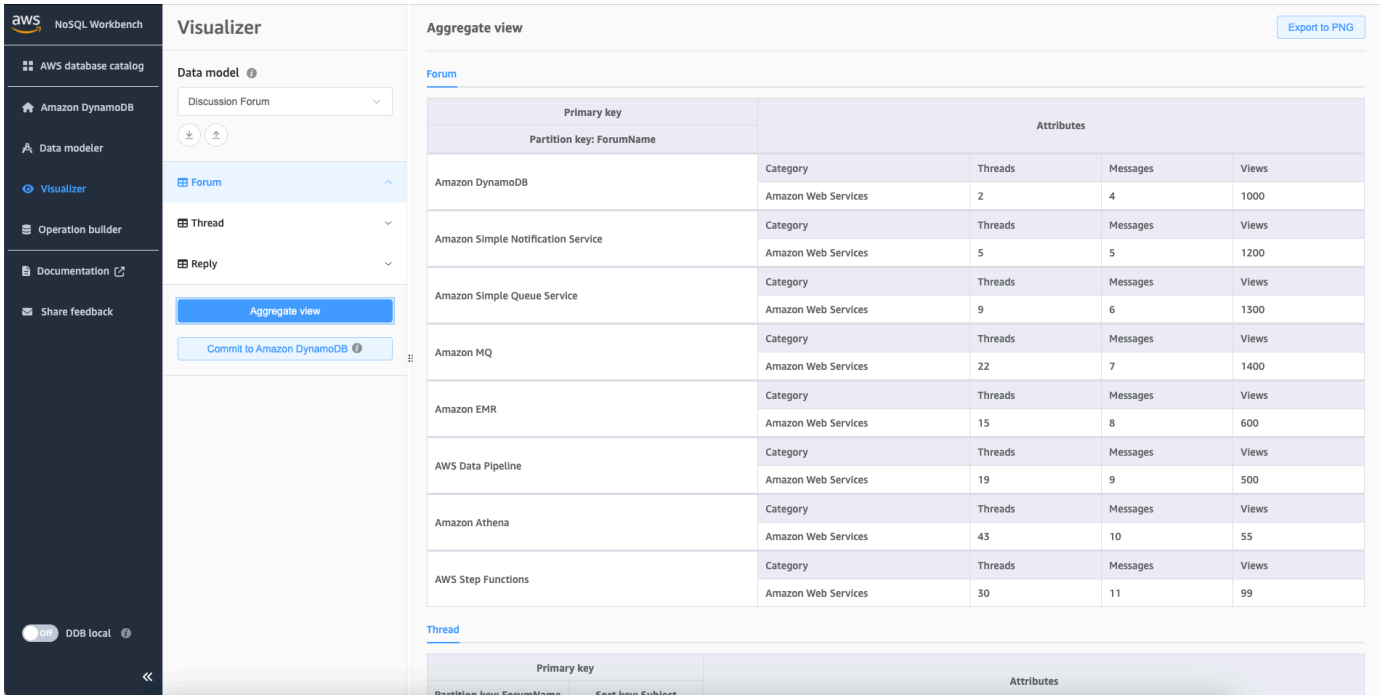
- Nom des colonnes de la table
- Exemples de données
- Tous les index secondaires globaux associés à la table. Les informations suivantes sont affichées pour chaque index :
  - Noms des colonnes de l'index
  - Exemples de données

Pour afficher les informations de toutes les tables

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).



2. Dans le visualiseur, choisissez Aggregate view (Vue d'agrégation).



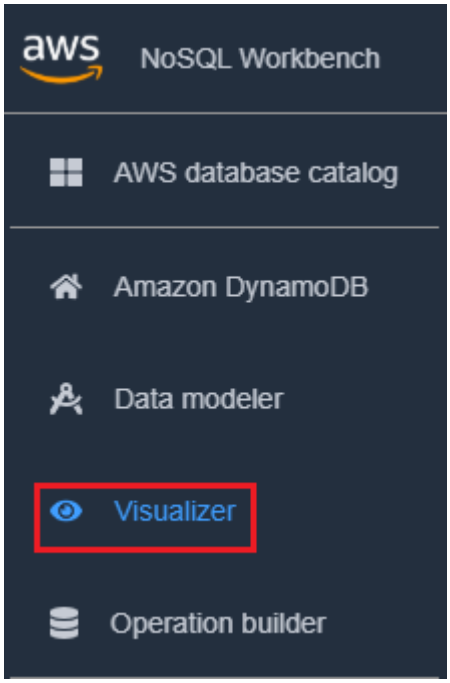
Validation d'un modèle de données dans DynamoDB

Lorsque vous êtes satisfait de votre modèle de données, vous pouvez valider le modèle dans Amazon DynamoDB.

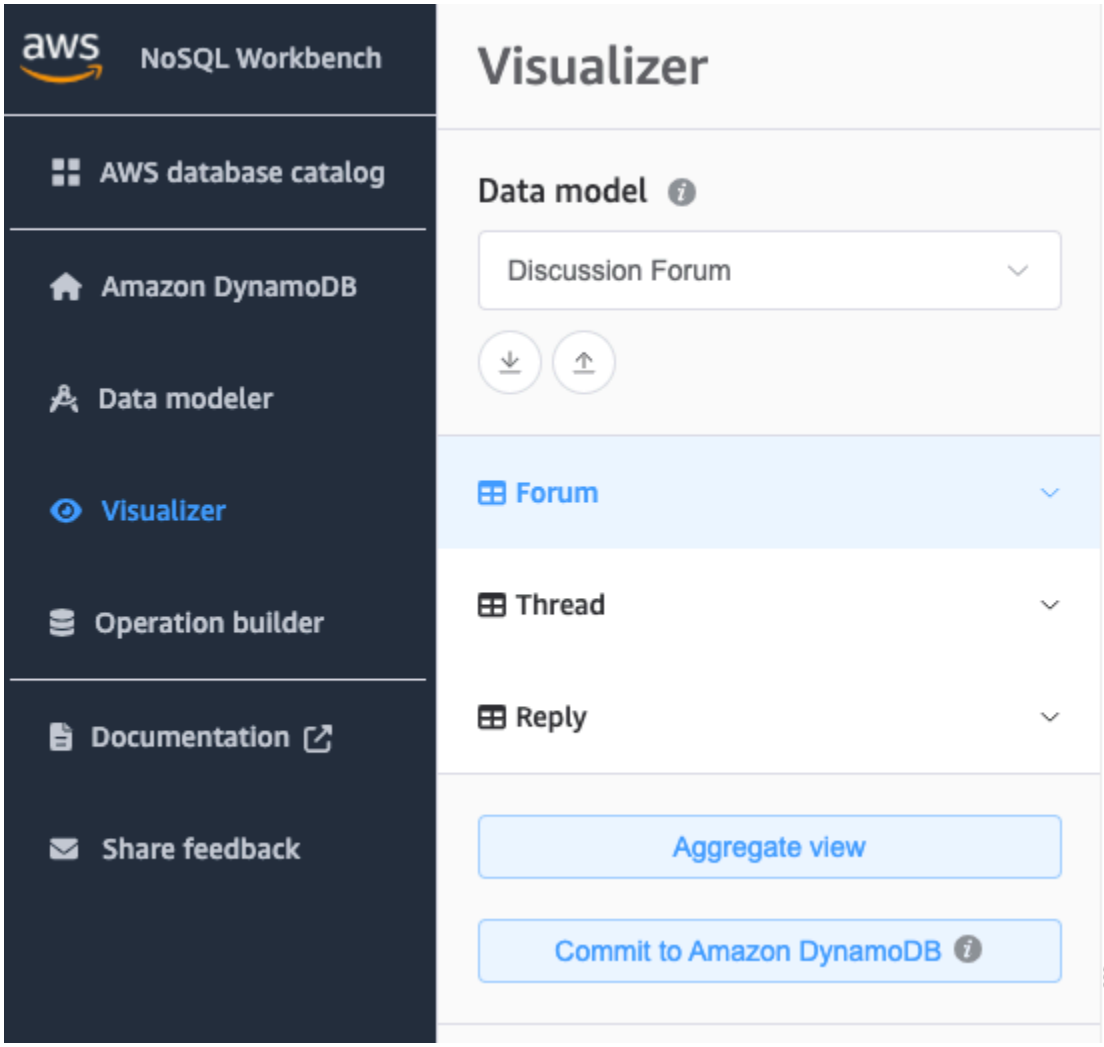
> **Note**
>
> - Cette action aboutit à la création des ressources côté serveur dans AWS pour les tables et les index secondaires globaux représentés dans le modèle de données.
> - Les tables sont créées avec les caractéristiques suivantes :
>   - L'auto scaling est défini sur 70 pour cent de l'utilisation cible.
>   - La capacité provisionnée est définie sur 5 unités de capacité de lecture et 5 unités de capacité d'écriture.
> - Les index secondaires globaux sont créés avec une capacité provisionnée de 10 unités de capacité de lecture et 5 unités de capacité d'écriture.

Pour valider le modèle de données dans DynamoDB

1. Dans le panneau de navigation de gauche, choisissez l'icône visualizer (visualiseur).



2. Choisissez Commit to DynamoDB (Valider dans DynamoDB).



3. Choisissez une connexion déjà existante, ou créez une nouvelle connexion en choisissant l'onglet Add new remote connection (Ajouter une nouvelle connexion distante).

   - Pour ajouter une nouvelle connexion, spécifiez les informations suivantes.
     - Alias du compte
     - Région AWS
     - ID de clé d'accès
     - Clé d'accès secrète

     Pour plus d'informations sur l'obtention des clés d'accès, consultez Obtention d'une clé d'accès AWS.
   - Le cas échéant, vous pouvez spécifier les options suivantes :
     - Le jeton de session

- L'ARN du rôle IAM
- Si vous ne voulez pas vous inscrire à un compte gratuit et que vous préférez utiliser DynamoDB Local (version téléchargeable) :
  1. Choisissez l'onglet Add a new DynamoDB local connection (Ajouter une nouvelle connexion à DynamoDB Local).
  2. Spécifiez le Nom de la connexion et le Port.

4. Choisissez Commit (Valider).

> **Note**
>
> Si vous avez installé DynamoDB local dans le cadre de la configuration de NoSQL Workbench, vous devez activer DynamoDB local en utilisant le bouton Serveur DynamoDB local en bas à gauche de l'écran NoSQL Workbench. Pour en savoir plus sur ce bouton, consultez Installer NoSQL Workbench pour DynamoDB.

Exploration des jeux de données et des opérations de création avec NoSQL Workbench

NoSQL Workbench pour Amazon DynamoDB fournit une interface graphique utilisateur riche pour développer et tester des requêtes. Vous pouvez utiliser le créateur d'opérations dans NoSQL Workbench pour consulter, explorer et interroger des ensembles de données. Le créateur d'opérations structurées prend en charge l'expression de projection ainsi que l'expression de condition et génère des exemples de code dans plusieurs langages. Vous pouvez cloner directement des tables d'un compte Amazon DynamoDB vers un autre dans différentes régions. Vous pouvez également cloner directement des tables entre DynamoDB local et un compte Amazon DynamoDB pour copier plus rapidement le schéma clé de votre table (et éventuellement le schéma et les éléments des GSI) entre vos environnements de développement. Vous pouvez enregistrer jusqu'à 50 opérations de données DynamoDB dans le créateur d'opérations.

Rubriques

- Se connecter à des ensembles de données en temps réel
- Création d'opérations complexes
- Clonage de tables avec NoSQL Workbench

- Exportation des données dans un fichier CSV

Se connecter à des ensembles de données en temps réel

Pour vous connecter à vos tables Amazon DynamoDB avec NoSQL Workbench, vous devez d'abord vous connecter à votre compte AWS.

Pour ajouter une connexion à votre base de données.

1. Dans NoSQL Workbench, dans le panneau de navigation sur la gauche, choisissez l'icône Operation builder (Créateur d'opérations).
2. Choisissez Add Connection (Ajouter une connexion).
3. Spécifiez les informations suivantes :

   - Connection name (Nom de la connexion)
   - Région AWS
   - ID de clé d'accès
   - Clé d'accès secrète

   Pour plus d'informations sur l'obtention des clés d'accès, consultez Obtention d'une clé d'accès AWS.

   Le cas échéant, vous pouvez spécifier les options suivantes :

   - Le jeton de session
   - L'ARN du rôle IAM
4. Choisissez Se connecter.

   Si vous ne voulez pas créer un compte gratuit et que vous préférez utiliser DynamoDB Local (version téléchargeable) :

   a. Choisissez l'onglet Local sur l'écran de connexion.
   b. Spécifiez les informations suivantes :

      - Connection name (Nom de la connexion)
      - Port

   c. Choisissez le bouton Connect (Se connecter).

> **Note**
>
> Pour vous connecter à DynamoDB local, lancez DynamoDB local manuellement à l'aide de votre terminal (consultez Déploiement de DynamoDB local sur votre ordinateur) ou lancez DynamoDB local directement à l'aide du bouton DDB local situé dans le menu de navigation de NoSQL Workbench. Assurez-vous que le port de connexion est le même que celui de votre port DynamoDB local.

5. Sur la connexion créée, choisissez Open (Ouvrir).

Une fois que vous êtes connecté à votre base de données DynamoDB, la liste des tables disponibles s'affiche dans le volet gauche. Choisissez l'une des tables pour retourner un exemple des données stockées dans la table.

Vous pouvez désormais exécuter des requêtes sur la table sélectionnée.

Pour exécuter des requêtes sur une table, consultez la section suivante sur la création d'opérations de création, Création d'opérations complexes.

Création d'opérations complexes

Le créateur d'opérations dans NoSQL Workbench pour Amazon DynamoDB fournit une interface visuelle dans laquelle vous pouvez effectuer des opérations complexes de plan de données. Il inclut la prise en charge des expressions de projection et de condition. Une fois que vous avez créé une opération, vous pouvez l'enregistrer en vue d'une utilisation ultérieure (vous pouvez enregistrer jusqu'à 50 opérations). Vous pouvez ensuite parcourir une liste de vos opérations de plan de données fréquemment utilisées dans le menu Saved Operations (Opérations enregistrées), et les utiliser pour remplir et créer automatiquement une nouvelle opération. Vous pouvez également choisir de générer un exemple de code pour ces opérations dans plusieurs langages.

NoSQL Workbench prend en charge la création d'instructions PartiQL pour DynamoDB, ce qui vous permet d'interagir avec DynamoDB à l'aide d'un langage de requête compatible SQL. Il prend également en charge la création d'opérations d'API CRUD DynamoDB.

Pour utiliser NoSQL Workbench afin de créer des opérations, dans le panneau de navigation de gauche, choisissez l'icône Operation builder (Créateur d'opérations).

Rubriques

- Création d'instructions PartiQL
- Création d'opérations d'API

Création d'instructions PartiQL

Pour utiliser NoSQL Workbench afin de créer des instructions PartiQL pour DynamoDB, choisissez Opérations PartiQL dans l'angle supérieur droit de NoSQL Workbench.

Vous pouvez créer les types d'instructions PartiQL suivants dans le créateur d'opérations.

Rubriques

- Instructions singleton
- Transactions
- Par lots

Instructions singleton

Pour exécuter ou générer du code pour une instruction PartiQL, procédez comme suit.

1. Choisissez Éditeur PartiQL en haut de la fenêtre.
2. Saisissez une Instruction PartiQL valide.
3. Si votre instruction utilise des paramètres :
    a. Choisissez Optional request parameters (Paramètres de demande facultatifs).
    b. Choisissez Add new parameters (Ajouter de nouveaux paramètres).
    c. Entrez le type d'attribut et la valeur.
    d. Si vous souhaitez ajouter des paramètres, répétez les étapes b et c.
4. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

    Sélectionnez votre langage souhaité dans les onglets affichés. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
5. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).
6. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save (Enregistrer). Entrez ensuite le nom de votre opération et choisissez Save (Enregistrer).

Transactions

Pour exécuter ou générer du code pour une transaction PartiQL, procédez comme suit.

1. Choisissez PartiQLTransaction dans la liste déroulante Autres opérations.
2. Choisissez Add a new statement (Ajouter une nouvelle instruction).
3. Saisissez une Instruction PartiQL valide.

> **Note**
>
> Les opérations de lecture et d'écriture ne sont pas prises en charge dans la même demande de transaction PartiQL. Une instruction SELECT ne peut pas figurer dans une même demande avec les instructions INSERT, UPDATE et DELETE. Pour plus d'informations, consultez Exécution de transactions avec PartiQL pour DynamoDB.

4. Si votre instruction utilise des paramètres
   a. Choisissez Optional request parameters (Paramètres de demande facultatifs).
   b. Choisissez Add new parameters (Ajouter de nouveaux paramètres).
   c. Entrez le type d'attribut et la valeur.
   d. Si vous souhaitez ajouter des paramètres, répétez les étapes b et c.
5. Si vous souhaitez ajouter d'autres instructions, répétez les étapes 2 à 4.
6. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Sélectionnez votre langage souhaité dans les onglets affichés. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
7. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).
8. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save (Enregistrer). Entrez ensuite le nom de votre opération et choisissez Save (Enregistrer).

Par lots

Pour exécuter ou générer du code pour un lot PartiQL, procédez comme suit.

1. Choisissez PartiQLBatch dans le menu déroulant Autres opérations.
2. Choisissez Add a new statement (Ajouter une nouvelle instruction).

3. Saisissez une Instruction PartiQL valide.

> **Note**
>
> Les opérations de lecture et d'écriture n'étant pas prises en charge dans une même demande par lot PartiQL, une instruction SELECT ne peut pas figurer dans la même demande que les instructions INSERT, UPDATE et DELETE. Les opérations d'écriture sur le même élément ne sont pas autorisées. Comme pour l'opération BatchGetItem, seules les opérations de lecture singleton sont prises en charge. Les opérations d'analyse et de requête ne sont pas prises en charge. Pour plus d'informations, consultez Exécution d'opérations par lot avec PartiQL pour DynamoDB.

4. Si votre instruction utilise des paramètres :

   a. Choisissez Optional request parameters (Paramètres de demande facultatifs).

   b. Choisissez Add new parameters (Ajouter de nouveaux paramètres).

   c. Entrez le type d'attribut et la valeur.

   d. Si vous souhaitez ajouter des paramètres, répétez les étapes b et c.

5. Si vous souhaitez ajouter d'autres instructions, répétez les étapes 2 à 4.

6. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Sélectionnez votre langage souhaité dans les onglets affichés. Vous pouvez désormais copier ce code et l'utiliser dans votre application.

7. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).

8. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save (Enregistrer). Entrez ensuite le nom de votre opération et choisissez Save (Enregistrer).

Création d'opérations d'API

Pour utiliser NoSQL Workbench afin de créer des API DynamoDB CRUD, sélectionnez Opérations sur une interface à gauche de l'interface utilisateur de NoSQL Workbench.

Sélectionnez ensuite Ouvrir et choisissez la connexion.

Vous pouvez effectuer les opérations suivantes dans le créateur d'opérations.

- Supprimer une table

- Créer une table
- Mettre à jour une table

- Mettre un élément
- Mettre à jour un élément
- Supprimer un élément
- Interrogation
- Analyser
- Transaction Obtenir des éléments
- Transaction des éléments en écriture

Supprimer une table

Pour exécuter une opération `Delete Table`, procédez comme suit.

1. Recherchez la table que vous souhaitez supprimer dans la section Tables.
2. Sélectionnez Supprimer la table dans le menu représentant des points de suspension.
3. Saisissez le nom de la table pour confirmer que vous voulez la supprimer.
4. Sélectionnez Supprimer.

Pour plus d'informations sur cette opération, consultez Supprimer une table dans la Référence d'API Amazon DynamoDB.

Suppression d'un index secondaire global

Pour exécuter une opération `Delete GSI`, procédez comme suit.

1. Recherchez le GSI d'une table que vous souhaitez supprimer dans la section Tables.
2. Sélectionnez Supprimer le GSI dans le menu représentant des points de suspension.
3. Saisissez le nom du GSI pour confirmer que vous voulez le supprimer.
4. Sélectionnez Supprimer.

Pour plus d'informations sur cette opération, consultez Supprimer une table dans la Référence d'API Amazon DynamoDB.

Create table

Pour exécuter une opération `Create Table`, procédez comme suit.

1. Cliquez sur l'icône + à côté de la section Tables.
2. Entrez le nom de table souhaité.
3. Créez une clé de partition.
4. Facultatif : créez une clé de tri.
5. Pour personnaliser les paramètres de capacité, décochez la case Utiliser les paramètres de capacité par défaut.
   - Vous pouvez désormais sélectionner l'une des options Alloué ou Capacité à la demande.

     Lorsque l'option Alloué est sélectionnée, vous pouvez définir des unités de capacité de lecture et d'écriture minimale et maximale. Vous pouvez également activer ou désactiver la mise à l'échelle automatique.
   - Si la table est définie sur le mode à la demande, vous ne pourrez pas spécifier de débit provisionné.
   - Si vous passez d'un débit en mode à la demande à un débit en mode provisionné, l'autoscaling est automatiquement appliqué à tous les GSI avec : min : 1, max : 10 ; cible : 70 %.
6. Sélectionnez Ignorer les GSI et créer pour créer cette table sans GSI. Vous avez aussi la possibilité de sélectionner Suivant pour créer un GSI avec cette nouvelle table.

Pour plus d'informations sur cette opération, consultez Créer une table dans la Référence d'API Amazon DynamoDB.

Création d'un GSI

Pour exécuter une opération `Create GSI`, procédez comme suit.

1. Trouvez une table à laquelle vous souhaitez ajouter un GSI.
2. Dans le menu représentant des points de suspension, sélectionnez Créer un GSI.
3. Donnez un nom à ce GSI sous Nom d'index.
4. Créez une clé de partition.
5. Facultatif : créez une clé de tri.
6. Choisissez une option de type de projection dans le menu déroulant.

7. Sélectionnez Créer un GSI.

Pour plus d'informations sur cette opération, consultez Créer une table dans la Référence d'API Amazon DynamoDB.

Mettre à jour une table

Pour mettre à jour les paramètres de capacité d'une table avec une opération `Update Table`, procédez comme suit.

1. Recherchez la table dont vous souhaitez mettre à jour les paramètres de capacité.
2. Dans le menu représentant des points de suspension, sélectionnez Mettre à jour les paramètres de capacité.
3. Sélectionnez Alloué ou Capacité à la demande.

   Lorsque l'option Alloué est sélectionnée, vous pouvez définir des unités de capacité de lecture et d'écriture minimale et maximale. Vous pouvez également activer ou désactiver la mise à l'échelle automatique.
4. Tâche de sélection Update (Mise à jour).

Pour plus d'informations sur cette opération, consultez Mettre à jour une table dans la Référence d'API Amazon DynamoDB.

Mise à jour d'un GSI

Pour mettre à jour les paramètres de capacité d'un GSI avec une opération `Update Table`, procédez comme suit.

> **Note**
>
> Par défaut, les index secondaires globaux héritent des paramètres de capacité de la table de base. Les index secondaires globaux ne peuvent avoir un mode de capacité différent que lorsque la table de base est en mode de capacité provisionnée. Lorsque vous créez un index secondaire global sur une table en mode approvisionné, vous devez spécifier des unités de capacité de lecture et d'écriture pour la charge de travail prévue sur cet index. Pour plus d'informations, consultez Considérations sur le débit alloué pour les index secondaires globaux.

1. Recherchez le GSI dont vous souhaitez mettre à jour les paramètres de capacité.
2. Dans le menu représentant des points de suspension, sélectionnez Mettre à jour les paramètres de capacité.
3. Vous pouvez désormais sélectionner l'une des options Alloué ou Capacité à la demande.

   Lorsque l'option Alloué est sélectionnée, vous pouvez définir des unités de capacité de lecture et d'écriture minimale et maximale. Vous pouvez également activer ou désactiver la mise à l'échelle automatique.
4. Tâche de sélection Update (Mise à jour).

Pour plus d'informations sur cette opération, consultez Mettre à jour une table dans la Référence d'API Amazon DynamoDB.

Mettre un élément

Pour créer un élément, utilisez l'opération `Put Item`. Pour exécuter ou générer du code pour une opération `Put Item`, procédez comme suit.

1. Recherchez la table dans laquelle vous souhaitez créer un élément.
2. Dans le menu déroulant Actions, sélectionnez Créer un élément.
3. Saisissez la valeur de la clé de partition.
4. Saisissez la valeur de la clé de tri, le cas échant.
5. Si vous souhaitez ajouter des attributs non-clés, procédez comme suit :

   a. Sélectionnez + Ajouter d'autres attributs.

   b. Spécifiez les champs Attribute name (Nom d'attribut), Type et Value (Valeur).
6. Si une expression de condition doit être satisfaite pour garantir le succès de l'opération `Put Item`, procédez comme suit :

   a. Choisissez Condition.

   b. Spécifiez le nom d'attribut, l'opérateur de comparaison, le type d'attribut et la valeur d'attribut.

   c. Si d'autres conditions sont nécessaires, choisissez de nouveau Condition.

   Pour plus d'informations, consultez Exemple de commande CLI d'expression de condition DynamoDB.

7. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Sélectionnez votre langage souhaité dans les onglets affichés. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
8. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).
9. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save operation (Enregistrer l'opération), entrez un nom pour votre opération, puis choisissez Save (Enregistrer).

Pour plus d'informations sur cette opération, consultez PutItem dans la Référence d'API Amazon DynamoDB.

Mettre à jour un élément

Pour exécuter ou générer du code pour une opération `Update Item`, procédez comme suit.

1. Recherchez la table dans laquelle vous souhaitez mettre à jour un élément.
2. Sélectionnez l'élément.
3. Saisissez le nom d'attribut et la valeur d'attribut pour l'expression sélectionnée.
4. Si vous souhaitez ajouter davantage d'expressions, choisissez une autre expression dans la liste déroulante Expression de mise à jour, puis sélectionnez l'icône +.
5. Si une expression de condition doit être satisfaite pour garantir le succès de l'opération `Update Item`, procédez comme suit :

   a. Choisissez Condition.

   b. Spécifiez le nom d'attribut, l'opérateur de comparaison, le type d'attribut et la valeur d'attribut.

   c. Si d'autres conditions sont nécessaires, choisissez de nouveau Condition.

   Pour plus d'informations, consultez Exemple de commande CLI d'expression de condition DynamoDB.
6. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Choisissez l'onglet du langage que vous souhaitez utiliser. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
7. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).

8. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save operation (Enregistrer l'opération), entrez un nom pour votre opération, puis choisissez Save (Enregistrer).

Pour plus d'informations sur cette opération, consultez UpdateItem dans la Référence d'API Amazon DynamoDB.

Supprimer un élément

Pour exécuter une opération `Delete Item`, procédez comme suit.

1. Recherchez la table dans laquelle vous souhaitez supprimer un élément.
2. Sélectionnez l'élément.
3. Dans le menu Actions, sélectionnez Supprimer un élément.
4. Sélectionnez Supprimer pour confirmer la suppression de l'élément.

Pour plus d'informations sur cette opération, consultez DeleteItem dans la Référence d'API Amazon DynamoDB.

Élément en double

Vous pouvez dupliquer un élément en créant un autre élément avec les mêmes attributs. Pour dupliquer un élément, procédez comme suit.

1. Recherchez la table dans laquelle vous souhaitez dupliquer un élément.
2. Sélectionnez l'élément.
3. Dans le menu Actions, sélectionnez Dupliquer l'élément.
4. Spécifiez une nouvelle clé de partition.
5. Spécifiez une nouvelle clé de tri (si nécessaire).
6. Sélectionnez Exécuter.

Pour plus d'informations sur cette opération, consultez DeleteItem dans la Référence d'API Amazon DynamoDB.

Requête

Pour exécuter ou générer du code pour une opération `Query`, procédez comme suit.

1. Sélectionnez Requête en haut de l'interface utilisateur de NoSQL Workbench.
2. Spécifiez la valeur de la clé de partition.
3. Si une clé de tri est nécessaire pour l'opération `Query`, procédez comme suit :

   a. Sélectionnez Sort key (Clé de tri).

   b. Spécifiez l'opérateur de comparaison et la valeur d'attribut.
4. Sélectionnez Requête pour exécuter cette opération de requête. Si d'autres options sont nécessaires, cochez la case Plus d'options et passez aux étapes suivantes.
5. Si tous les attributs ne doivent pas être retournés avec le résultat de l'opération, sélectionnez Projection expression (Expression de projection).
6. Choisissez l'icône +.
7. Saisissez l'attribut à retourner avec le résultat de la requête.
8. Si davantage d'attributs sont nécessaires, choisissez le +.
9. Si une expression de condition doit être satisfaite pour garantir le succès de l'opération `Query`, procédez comme suit :

   a. Choisissez Condition.

   b. Spécifiez le nom d'attribut, l'opérateur de comparaison, le type d'attribut et la valeur d'attribut.

   c. Si d'autres conditions sont nécessaires, choisissez de nouveau Condition.

   Pour plus d'informations, consultez Exemple de commande CLI d'expression de condition DynamoDB.
10. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

    Choisissez l'onglet du langage que vous souhaitez utiliser. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
11. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).
12. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save operation (Enregistrer l'opération), entrez un nom pour votre opération, puis choisissez Save (Enregistrer).

Pour plus d'informations sur cette opération, consultez Query dans la Référence d'API Amazon DynamoDB.

Analyser

Pour exécuter ou générer du code pour une opération `Scan`, procédez comme suit.

1. Sélectionnez Analyser en haut de l'interface utilisateur de NoSQL Workbench.
2. Cliquez sur le bouton Analyser pour effectuer cette opération d'analyse de base. Si d'autres options sont nécessaires, cochez la case Plus d'options et passez aux étapes suivantes.
3. Spécifiez un nom d'attribut pour filtrer les résultats de l'analyse.
4. Si tous les attributs ne doivent pas être retournés avec le résultat de l'opération, sélectionnez Projection expression (Expression de projection).
5. Si une expression de condition doit être satisfaite pour garantir le succès de l'opération Scan, procédez comme suit :

   a. Choisissez Condition.

   b. Spécifiez le nom d'attribut, l'opérateur de comparaison, le type d'attribut et la valeur d'attribut.

   c. Si d'autres conditions sont nécessaires, choisissez de nouveau Condition.

   Pour plus d'informations, consultez Exemple de commande CLI d'expression de condition DynamoDB.
6. Si vous souhaitez générer un code, choisissez Generate code (Générer un code).

   Choisissez l'onglet du langage que vous souhaitez utiliser. Vous pouvez désormais copier ce code et l'utiliser dans votre application.
7. Si vous souhaitez que l'opération soit exécutée immédiatement, choisissez Run (Exécuter).
8. Si vous souhaitez enregistrer cette opération en vue d'une utilisation ultérieure, choisissez Save operation (Enregistrer l'opération), entrez un nom pour votre opération, puis choisissez Save (Enregistrer).

TransactGetItems

Pour exécuter ou générer du code pour une opération `TransactGetItems`, procédez comme suit.

1. Dans le menu déroulant Autres opérations en haut de l'interface utilisateur de NoSQL Workbench, sélectionnez TransactGetItems.
2. Cliquez sur l'icône + à côté de TransactGetItem.

3. Spécifiez une clé de partition.
4. Spécifiez une clé de tri (si nécessaire).
5. Sélectionnez Exécuter pour effectuer l'opération, Enregistrer l'opération pour l'enregistrer ou Générer le code pour générer le code correspondant.

Pour de plus amples informations sur les transactions, consultez Transactions Amazon DynamoDB.

TransactWriteItems

Pour exécuter ou générer du code pour une opération `TransactWriteItems`, procédez comme suit.

1. Dans le menu déroulant Autres opérations en haut de l'interface utilisateur de NoSQL Workbench, sélectionnez TransactWriteItems.
2. Choisissez une opération dans le menu déroulant Actions.
3. Cliquez sur l'icône + à côté de TransactWriteItem.
4. Dans le menu déroulant Actions, choisissez l'opération que vous souhaitez effectuer.

   - Pour `DeleteItem`, suivez les instructions pour l'opération Supprimer un élément.
   - Pour `PutItem`, suivez les instructions pour l'opération Mettre un élément.
   - Pour `UpdateItem`, suivez les instructions pour l'opération Mettre à jour un élément.

   Pour modifier l'ordre des actions, choisissez une action dans la liste à gauche, puis utilisez la flèche du haut ou du bas pour la déplacer dans la liste.

   Pour supprimer une action, choisissez l'action dans la liste, puis choisissez l'icône supprimer (corbeille).
5. Sélectionnez Exécuter pour effectuer l'opération, Enregistrer l'opération pour l'enregistrer ou Générer le code pour générer le code correspondant.

Pour de plus amples informations sur les transactions, consultez Transactions Amazon DynamoDB.

Clonage de tables avec NoSQL Workbench

Le clonage des tables permet de copier le schéma de clés d'une table (et éventuellement le schéma GSI et les éléments) entre vos environnements de développement. Vous pouvez cloner une table

entre DynamoDB local et un compte Amazon DynamoDB, voire cloner une table d'un compte à un autre dans différentes régions pour accélérer les tests.

Pour cloner une table

1. Dans le Créateur d'opérations, sélectionnez votre connexion et votre région (la sélection de la région n'est pas disponible pour DynamoDB local).
2. Une fois connecté à DynamoDB, parcourez vos tables et sélectionnez celle que vous souhaitez cloner.
3. Dans le menu représentant des points de suspension, sélectionnez l'option Cloner.
4. Entrez les détails de la destination du clone :
    a. Sélectionnez une connexion.
    b. Sélectionnez une région (la région n'est pas disponible pour DynamoDB local).
    c. Entrez un nom pour la nouvelle table.
    d. Choisissez une option de clonage :
        i. L'option Schéma de clés est sélectionnée par défaut et ne peut pas être désélectionnée. Par défaut, le clonage d'une table copie votre clé primaire et votre clé de tri si elles sont disponibles.
        ii. L'option Schéma GSI est sélectionnée par défaut si votre table à cloner possède un GSI. Le clonage d'une table copie la clé primaire et la clé de tri de votre GSI si elles sont disponibles. Vous avez la possibilité de désélectionner l'option Schéma GSI pour ignorer le clonage du schéma GSI. Le clonage d'une table copie les paramètres de capacité de votre table de base en tant que paramètres de capacité du GSI. Vous pouvez utiliser l'opération `UpdateTable` dans le créateur d'opérations pour mettre à jour le paramètre de capacité du GSI de la table une fois le clonage terminé.
5. Entrez le nombre d'éléments à cloner. Pour cloner uniquement le schéma de clés et éventuellement le schéma GSI, vous pouvez conserver la valeur 0 pour Éléments à cloner. Le nombre maximum d'éléments pouvant être clonés est de 5 000.
6. Choisissez un mode de capacité :
    a. Le mode à la demande est sélectionné par défaut. DynamoDB à la demande offre une tarification de « paiement par demande » où vous ne payez que les demandes de lecture et d'écriture que vous exécutez. Pour en savoir plus, consultez DynamoDB On-demand mode.

b. Le mode provisionné vous permet de spécifier le nombre de lectures et d'écritures par seconde nécessaires pour votre application. Vous pouvez recourir à l'autoscaling pour ajuster automatiquement la capacité allouée de votre table en fonction de l'évolution du trafic. Pour en savoir plus, consultez Mode provisionné DynamoDB.

7. Sélectionnez Cloner pour commencer le clonage.
8. Le processus de clonage s'exécute en arrière-plan. L'onglet Créateur d'opérations affiche une notification en cas de modification de l'état de la table de clonage. Vous pouvez accéder à cet état en sélectionnant l'onglet Créateur d'opérations, puis en sélectionnant le bouton fléché. Le bouton fléché se trouve sur le widget d'état de la table de clonage situé en bas de la barre latérale du menu.

Exportation des données dans un fichier CSV

Vous pouvez exporter les résultats d'une requête depuis Operation Builder vers un fichier CSV. Cela vous permet de charger les données dans une feuille de calcul ou de les traiter à l'aide de votre langage de programmation préféré.

Exportation vers CSV

1. Dans Operation Builder, exécutez une opération de votre choix, telle qu'une analyse ou une requête.

> **Note**
>
> - Vous ne pouvez exporter que les résultats des opérations d'API de lecture et des instructions PartiQL vers un fichier CSV. Vous ne pouvez pas exporter les résultats des instructions de lecture des transactions.
> - À l'heure actuelle, vous pouvez exporter les résultats vers un fichier CSV page par page. S'il existe plusieurs pages de résultats, vous devez exporter chaque page individuellement.

2. Sélectionnez les éléments que vous souhaitez exporter dans les résultats.
3. Dans le menu déroulant Actions, choisissez Exporter au format CSV.
4. Choisissez un nom de fichier et un emplacement pour votre fichier CVS et sélectionnez Enregistrer.

Exemples de modèle de données pour NoSQL Workbench

La page d'accueil du modélisateur et du visualiseur affiche un certain nombre d'exemples de modèle livrés avec NoSQL Workbench. Cette section décrit ces modèles et leurs utilisations potentielles.

Rubriques

- Modèle de données d'employé
- Modèle de données de forum de discussion
- Modèle de données de bibliothèque musicale
- Modèle de données de station de ski
- Modèle de données d'offres de carte de crédit
- Modèle de données de signets

Modèle de données d'employé

Ce modèle de données est un modèle d'introduction. Il représente les détails de base d'un employé, tels qu'un alias unique, son prénom, son nom de famille, son intitulé de poste, son responsable et ses compétences.

Ce modèle de données décrit quelques techniques telles que la gestion d'attributs complexes (par exemple, le fait d'avoir plusieurs compétences). Ce modèle est également un exemple de relation de type « un à plusieurs » entre le responsable et les employés qui dépendent de celui-ci. Cette relation a été définie par l'index secondaire DirectReports.

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Récupération d'un enregistrement d'employé à l'aide de l'alias de connexion de l'employé. Cette récupération est facilitée par une table appelée `Employee`.
- Recherchez les employés par nom. Cette recherche est facilitée par l'index secondaire global de la table Employee appelé `Name`.
- Récupération de tous les rapports directs d'un responsable à l'aide de l'alias de connexion du responsable. Cette récupération est facilitée par l'index secondaire global de la table Employee appelé `DirectReports`.

Modèle de données de forum de discussion

Ce modèle de données représente un forum de discussion. En utilisant ce modèle, les clients peuvent dialoguer avec la communauté de développeurs, poser des questions et répondre aux messages des autres clients. Chaque service AWS a un forum dédié. N'importe qui peut démarrer un nouveau fil de discussion en publiant un message dans un forum, chaque fil recevant un nombre quelconque de réponses.

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Récupération d'un enregistrement de forum à l'aide du nom du forum. Cette récupération est facilitée par une table appelée `Forum`.
- Récupération d'un fil spécifique ou de tous les fils d'un forum. Cette récupération est facilitée par une table appelée `Thread`.
- Recherchez des réponses à l'aide de l'adresse e-mail de l'utilisateur à l'origine de la publication. Cette recherche est facilitée par l'index secondaire global de la table Reply appelé `PostedBy-Message-Index`.

Modèle de données de bibliothèque musicale

Ce modèle de données représente une bibliothèque musicale qui possède une importante collection de chansons et met en avant les chansons les plus téléchargées en temps quasi réel.

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Récupération d'un enregistrement de chanson. Cette récupération est facilitée par une table appelée `Songs`.
- Récupération d'un enregistrement de téléchargement spécifique ou de tous les enregistrements de téléchargement d'une chanson. Cette récupération est facilitée par une table appelée `Songs`.
- Récupération d'un enregistrement mensuel spécifique du nombre de téléchargements ou de tous les enregistrements mensuels du nombre de téléchargements pour une chanson. Cette récupération est facilitée par une table appelée `Song`.
- Récupération de tous les enregistrements (y compris les enregistrements de chanson, les enregistrements de téléchargement et les enregistrements mensuels du nombre de téléchargements) pour une chanson. Cette récupération est facilitée par une table appelée `Songs`.
- Recherchez les chansons les plus téléchargées. Cette recherche est facilitée par l'index secondaire global de la table Songs appelé `DownloadsByMonth`.

Modèle de données de station de ski

Ce modèle de données représente une station de ski qui dispose d'une importante collection de données pour chaque remontée mécanique recueillies quotidiennement.

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Récupération de toutes les données (dynamiques et statiques) pour une remontée mécanique ou l'ensemble d'une station donnée. Cette récupération est facilitée par une table appelée `SkiLifts`.
- Récupération de toutes les données dynamiques (y compris les utilisateurs d'une remontée, la couverture de neige, le danger d'avalanche et l'état de la remontée) pour une remontée mécanique ou l'ensemble de la station à une date précise. Cette récupération est facilitée par une table appelée `SkiLifts`.
- Récupération de toutes les données statiques (y compris le fait de savoir si la remontée est conçue pour des utilisateurs expérimentés uniquement, le dénivelé couvert par la remontée et le temps nécessaire pour la montée) pour une remontée mécanique spécifique, facilitée par une table appelée `SkiLifts`.
- Récupération de la date des données enregistrées pour une remontée mécanique spécifique ou pour l'ensemble de la station, triée par le nombre d'utilisateurs uniques. Cette récupération est facilitée par l'index secondaire global de la table SkiLifts appelé `SkiLiftsByRiders`.

Modèle de données d'offres de carte de crédit

Ce modèle de données est utilisé par une application d'offres de carte de crédit.

Un fournisseur de cartes de crédit produit des offres au fil du temps. Ces offres comprennent des transferts de solde sans frais, des augmentations de limite de crédit, des baisses de taux d'intérêt, des remboursements et des miles aériens. Lorsqu'un client accepte ou refuse ces offres, le statut de l'offre respective est mis à jour en conséquence.

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Récupération des enregistrements de compte à l'aide de `AccountId`. Cette récupération est facilitée par la table principale.
- Récupération de tous les comptes avec quelques éléments projetés. Cette récupération est facilitée par l'index secondaire `AccountIndex`.

- Récupération des comptes et tous les enregistrements d'offre associés à ces comptes à l’aide de `AccountId`. Cette récupération est facilitée par la table principale.
- Récupération des comptes et des enregistrements d'offre spécifique associés à ces comptes à l’aide de `AccountId` et `OfferId`. Cette récupération est facilitée par la table principale.
- Récupération de tous les enregistrements d'offre `ACCEPTED/DECLINED` avec `OfferType` spécifique associés à des comptes à l'aide de `AccountId`, `OfferType` et `Status`. Cette récupération est facilitée par l'index secondaire `GSI1`.
- Récupération des offres et des enregistrements d'élément d'offre associés à l'aide de `OfferId`. Cette récupération est facilitée par la table principale.

Modèle de données de signets

Ce modèle de données permet de stocker des signets pour des clients.

Un client peut avoir de nombreux signets et un signet peut appartenir à de nombreux clients. Ce modèle de données représente une relation de type « plusieurs à plusieurs ».

Les modèles d'accès facilités par ce modèle de données sont les suivants :

- Une requête unique par `customerId` peut désormais renvoyer des données client ainsi que des signets.
- Un index de requête `ByEmail` renvoie les données client par adresse e-mail. Notez que les signets ne sont pas récupérés par cet index.
- Un index de requête `ByUrl` permet d’obtenir les données de signets par URL. Notez que CustomerID constitue la clé de tri pour l'index, car la même URL peut être marquée par plusieurs clients.
- Un index de requête `ByCustomerFolder` permet d’obtenir des signets par dossier pour chaque client.

Historique de version pour NoSQL Workbench

Le tableau ci-après décrit les modifications importantes apportées à chaque version de l’outil client NoSQL Workbench.

Version	Modification	Description	Date
3.13.5	Le mode de capacité pour les paramètre s de table par défaut est désormais à la demande	Lorsque vous créez une table avec les paramètres par défaut, DynamoDB crée une table qui utilise le mode de capacité à la demande au lieu du mode de capacité provisionnée.	24 février 2025
3.13.0	Améliorations du créateur d'opérations NoSQL Workbench	NoSQL Workbench inclut désormais la prise en charge native du mode sombre. Améliorat ion des opération s sur les tables et les éléments dans le créateur d'opérati ons. Les informati ons sur les résultats des éléments et les demandes du créateur d'opérations sont disponibles au format JSON.	24 avril 2024
3.12.0	Clonage de tables avec NoSQL Workbench et affichage de la capacité consommée	Vous pouvez désormais cloner des tables entre DynamoDB local et un compte de service Web DynamoDB ou entre des comptes	26 février 2024

Version	Modification	Description	Date
		de service Web DynamoDB pour accélérer les itérations de développement. Affichez les RCU ou WCU consommées après l'exécution d'une opération à l'aide du créateur d'opérations. Nous avons résolu le problème d'écrasement des données lors de l'importation à partir d'un fichier CSV.	
3.11.0	Améliorations de DynamoDB Local	Vous pouvez désormais spécifier le port lors du lancement de l'instance DynamoDB local intégrée. NoSQL Workbench peut désormais être installé sur Windows sans droits d'administrateur. Nous avons mis à jour les modèles de données.	17 janvier 2024

Version	Modification	Description	Date
3.10.0	Prise en charge native pour le silicium Apple	NoSQL Workbench inclut désormais une prise en charge native pour Mac avec le silicium Apple. Vous pouvez désormais configurer le format de génération des données d'échanti llonnage pour les attributs de type `Number`.	5 décembre 2023
3.9.0	Améliorations du modélisateur de données	Le visualiseur prend désormais en charge la validation des modèles de données dans DynamoDB Local avec la possibili té de remplacer les tables existantes.	3 novembre 2023
3.8.0	Génération de données d'exemple	NoSQL Workbench prend désormais en charge la générati o n de données d'exemple pour vos modèles de données DynamoDB.	25 septembre 2023

Version	Modification	Description	Date
3.6.0	Améliorations apportées au créateur d'opérations	Améliorations de la gestion des connexions dans le créateur d'opérations. Les noms d'attributs dans Data Modeler peuvent désormais être modifiés sans suppression de données. Autres correctifs de bogues.	11 avril 2023
3.5.0	Prise en charge des nouvelles régions AWS	NoSQL Workbench prend désormais en charge les régions ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 et me-west-1.	23 février 2023
3.4.0	Prise en charge de DynamoDB local	NoSQL Workbench prend désormais en charge DynamoDB local dans le cadre du processus d'installation.	6 décembre 2022
3.3.0	Support des opérations de plans de contrôle	Operation Builder prend désormais en charge les opérations de contrôle.	1 juin 2022

Version	Modification	Description	Date
3.2.0	Importation et exportation de fichiers CSV	Vous pouvez désormais importer des exemples de données à partir d'un fichier CSV dans l'outil Visualize r. Vous pourrez également exporter les résultats d'une requête Operation Builder vers un fichier CSV.	11 octobre 2021
3.1.0	Enregistrer des opérations	Le créateur d'opérati ons dans NoSQL Workbench prend désormais en charge les opérations d'enregistrement en vue d'une utilisation ultérieure.	12 Juillet 2021
3.0.0	Paramètres de capacité et importati on/exportation CloudFormation	NoSQL Workbench pour Amazon DynamoDB prend désormais en charge la spécification d'un mode de capacité de lecture/écriture pour les tables, et peut importer et exporter des modèles de données au format CloudFormation.	21 avril 2021

Version	Modification	Description	Date
2.2.0	Prise en charge de PartiQL	NoSQL Workbench pour Amazon DynamoDB ajoute la prise en charge de la création d'instructions PartiQL pour DynamoDB.	4 décembre 2020
1.1.0	Prise en charge de Linux.	NoSQL Workbench pour Amazon DynamoDB est pris en charge sur Linux Ubuntu, Fedora et Debian.	4 mai 2020
1.0.0	NoSQL Workbench for Amazon DynamoDB – GA.	NoSQL Workbench pour Amazon DynamoDB est généralement disponible.	2 mars 2020
0.4.1	Prise en charge des rôles IAM et des informations d'identification de sécurité temporaires.	NoSQL Workbench pour Amazon DynamoDB ajoute la prise en charge des rôles et des informations d'identification de sécurité temporaires Gestion des identités et des accès AWS (IAM).	19 décembre 2019

Version	Modification	Description	Date
0.3.1	Prise en charge de DynamoDB local (version téléchargeable)	NoSQL Workbench prend désormais en charge la connexion à DynamoDB local (version téléchargeable) pour concevoir, créer, interroger et gérer des tables DynamoDB.	8 novembre 2019
0.2.1	Version préliminaire de NoSQL Workbench.	Il s'agit de la version initiale de NoSQL Workbench pour DynamoDB. Utilisez NoSQL Workbench pour concevoir, créer, interroger et gérer des tables DynamoDB.	16 septembre 2019

Sauvegarde et restauration pour DynamoDB

DynamoDB propose des sauvegardes à la demande et de reprise ponctuelle (PITR) pour protéger vos données DynamoDB en cas de sinistre. Il propose également un archivage des données pour une conservation à long terme. Vous pouvez sauvegarder des tables de quelques méga-octets à des centaines de téra-octets de données, sans impact sur les performances et la disponibilité de vos applications de production. Toutes les sauvegardes sont automatiquement chiffrées, cataloguées et facilement détectables.

Avec les sauvegardes à la demande, vous pouvez créer une sauvegarde instantanée de votre table que DynamoDB stocke et gère. Vous êtes facturé en fonction de la taille et de la durée de vos sauvegardes. Grâce à la sauvegarde à la demande, vous pouvez restaurer l'intégralité de votre table DynamoDB à l'état dans lequel elle était au moment de la création de la sauvegarde.

Deux options sont disponibles pour créer et gérer des sauvegardes DynamoDB à la demande :

- DynamoDB
- AWS Backup

Vous pouvez utiliser la fonction de sauvegarde DynamoDB à la demande pour créer des sauvegardes complètes de vos tables pour l'archivage et la rétention à long terme, à des fins de conformité réglementaire. Vous pouvez sauvegarder et restaurer les données de vos tables à tout moment grâce à un seul clic dans la AWS Management Console ou à un seul appel d'API.

Les sauvegardes de reprise ponctuelle (PITR) sont entièrement gérées par DynamoDB et fournissent jusqu'à 35 jours de points de restauration à une granularité par seconde. Pour utiliser la reprise ponctuelle, qui est une sauvegarde continue, activez la reprise ponctuelle (PITR) sur votre table DynamoDB. Vous êtes facturé en fonction de la taille de votre table DynamoDB pour la durée pendant laquelle la reprise ponctuelle (PITR) est activée sur la table. L'activation de la reprise ponctuelle (PITR) sur votre table DynamoDB sauvegarde en permanence vos données. Cela vous permet de restaurer votre table à un moment précis pendant la période de restauration de reprise ponctuelle (PITR) en créant une nouvelle table DynamoDB avec l'état exact de votre table d'origine à ce moment-là.

La restauration à un instant dans le passé permet de protéger vos tables DynamoDB contre les opérations d'écriture ou de suppression accidentelles. Grâce à la restauration à un instant dans le passé, vous n'avez plus à vous soucier de la création, de la maintenance ou de la planification des

sauvegardes à la demande. Par exemple, imaginez qu'un script de test écrive accidentellement sur une table DynamoDB de production.

Grâce à la reprise ponctuelle, vous pouvez restaurer cette table à n'importe quel instant dans le passé au cours des 35 derniers jours. Vous pouvez définir la période de restauration sur une valeur comprise entre 1 et 35 jours. Une fois la reprise ponctuelle activée, vous pouvez restaurer à n'importe quel moment à partir de cinq minutes avant l'heure actuelle jusqu'à la période de restauration configurée. DynamoDB conserve des sauvegardes incrémentielles de votre table.

En outre, les opérations de restauration à un instant dans le passé n'affectent pas les performances ou les latences des API.

Vous pouvez restaurer une table DynamoDB à un instant dans le passé à l'aide de l'AWS Management Console, de l'AWS Command Line Interface (AWS CLI) ou de l'API DynamoDB. Le processus de restauration à un instant dans le passé restaure une nouvelle table.

Pour plus d'informations sur la disponibilité et la tarification d'une région AWS, consultez Tarification d'Amazon DynamoDB.

> **Note**
>
> - Le balisage et le contrôle d'accès par attributs (ABAC) ne sont pas pris en charge pour les sauvegardes DynamoDB. Pour utiliser l'ABAC avec des sauvegardes, nous vous recommandons d'utiliser AWS Backup.
> - Les balises ne sont pas conservées dans les tables restaurées. Vous devez ajouter des balises aux tables restaurées avant de pouvoir utiliser des conditions basées sur des balises dans vos politiques.

La vidéo suivante présente le concept de sauvegarde et de restauration et décrit en détail la restauration à un instant dans le passé.

Sauvegarde et restauration

Rubriques

- Sauvegardes ponctuelles pour DynamoDB
- Utilisation de la sauvegarde et de la restauration à la demande de DynamoDB
- Présentation de la facturation des sauvegardes pour Amazon DynamoDB

- Restauration d'une table dans DynamoDB
- Utilisation de AWS Backup avec DynamoDB

Sauvegardes ponctuelles pour DynamoDB

La reprise ponctuelle (PITR) d'Amazon DynamoDB fournit des sauvegardes continues des données de votre table DynamoDB. Les sauvegardes de reprise ponctuelle (PITR) sont entièrement gérées par DynamoDB et fournissent jusqu'à 35 jours de points de restauration à une granularité par seconde. Grâce à la reprise ponctuelle, vous n'avez plus à vous soucier de la création, de la maintenance ou de la planification des sauvegardes à la demande. Cette section présente le fonctionnement du processus dans DynamoDB.

Avant de commencer

Avant d'activer la restauration à un instant dans le passé sur une table Amazon DynamoDB, vérifiez les points suivants :

- La définition de `RecoveryPeriodinDays` vous permet de raccourcir la période pendant laquelle les sauvegardes continues sont effectuées. Par défaut, la valeur de `RecoveryPeriodinDays` est 35. Toutefois, vous pouvez le définir sur une valeur comprise entre 1 et 35. Le raccourcissement de `RecoveryPeriodinDays` n'a aucun impact sur la tarification du PITR, car le prix est basé sur la taille de la table et sur les indices secondaires locaux.
- Si vous désactivez la restauration à un instant dans le passé pour la réactiver ultérieurement sur une table, vous réinitialisez l'heure de début à laquelle vous pouvez restaurer cette table. Ainsi, vous pouvez uniquement restaurer instantanément cette table à l'aide de `LatestRestorableDateTime`.
- Vous pouvez activer la restauration à un instant dans le passé sur chaque réplica local d'une table globale. Lorsque vous restaurez la table, la sauvegarde restaure une table indépendante ne faisant pas partie de la table globale. Si vous utilisez la version 2019.11.21 (actuelle) des tables globales, vous pouvez créer une nouvelle table globale à partir de la table restaurée. Pour plus d'informations, consultez Fonctionnement des tables globales DynamoDB.
- Vous pouvez également restaurer les données de votre table DynamoDB dans toutes les régions AWS afin que la table restaurée soit créée dans une région différente de celle où réside la table source. Vous pouvez effectuer des restaurations entre régions commerciales AWS, régions AWS Chine et les régions AWS GovCloud (US). Vous payez uniquement pour les données que

vous transférez hors de la région source et pour la restauration vers une nouvelle table dans la région de destination.

- AWS CloudTrail enregistre toutes les actions de console et d'API pour la restauration à un instant dans le passé afin d'activer la journalisation, la surveillance en continu et l'audit. Pour plus d'informations, consultez Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail.

Rubriques

- Activation de la reprise ponctuelle pour DynamoDB

Activation de la reprise ponctuelle pour DynamoDB

La reprise ponctuelle (PITR) d'Amazon DynamoDB fournit des sauvegardes automatiques des données de votre table DynamoDB. Cette section présente le fonctionnement du processus dans DynamoDB.

> **Note**
>
> DynamoDB facture la reprise ponctuelle (PITR) en fonction de la taille de chaque table DynamoDB, y compris les données de la table et les index secondaires locaux. La modification du délai de reprise (par exemple, de 35 jours à 1 jour) ne réduit pas le prix. Le coût reste le même quelle que soit la période de reprise choisie. La période de reprise maximale configurée n'a aucune incidence sur le prix qui vous est facturé pour l'activation de la PITR. Pour déterminer vos frais de sauvegarde, DynamoDB surveille en permanence la taille des tables sur lesquelles la PITR est activée. L'utilisation de la PITR vous est facturée jusqu'à ce que vous la désactiviez pour chaque table.

Rubriques

- Activation de la reprise ponctuelle (PITR)
- Activation de la reprise ponctuelle (PITR)
- Activation de la reprise ponctuelle (PITR) (AWS CLI)
- Activation de la reprise ponctuelle (PITR) (CloudFormation)
- Activation de la reprise ponctuelle (PITR) (API)
- Période de reprise

- Modification de la reprise ponctuelle (PITR)
- Suppression d'une table avec l'activation de la reprise ponctuelle (PITR)

Activation de la reprise ponctuelle (PITR)

Vous pouvez activer la reprise ponctuelle (PITR) à l'aide de la AWS Management Console, de l'AWS Command Line Interface (AWS CLI) ou de l'API DynamoDB. Une fois activée, la reprise ponctuelle (PITR) fournit des sauvegardes continues jusqu'à ce que vous la désactiviez.

Après avoir activé la reprise ponctuelle (PITR), vous pouvez restaurer à n'importe quel instant dans le passé, entre le `EarliestRestorableDateTime` et le `LatestRestorableDateTime`. `LatestRestorableDateTime` précède généralement de 5 minutes l'heure actuelle. Pour plus d'informations, consultez Restauration d'une table DynamoDB à un instant dans le passé.

> **Note**
>
> Le processus de reprise ponctuelle (PITR) restaure toujours une nouvelle table.

Activation de la reprise ponctuelle (PITR)

Pour activer la reprise ponctuelle (PITR) à l'aide de la console DynamoDB

1. Accédez à la console DynamoDB.
2. Choisissez Tables dans le menu de navigation de gauche et sélectionnez votre table DynamoDB.
3. Dans l'onglet Sauvegardes, pour l'option Reprise ponctuelle (PITR), choisissez Modifier.
4. Choisissez Activer la reprise ponctuelle (PITR).
5. Choisissez une valeur comprise entre 1 et 35 pour la période de reprise de votre sauvegarde. Cela indique la durée maximale pendant laquelle la sauvegarde continue est récupérable.

Activation de la reprise ponctuelle (PITR) (AWS CLI)

> **Note**
>
> Si vous recevez des erreurs lors de l'exécution de commandes AWS CLI, consultez Troubleshoot AWS CLI errors. Assurez-vous d'utiliser la version la plus récente de la AWS CLI.

Exécutez la commande update-continuous-backups avec le paramètre point-in-time-recovery-specification activé :

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Activation de la reprise ponctuelle (PITR) (CloudFormation)

Utilisez la ressource AWS::DynamoDB::Table avec la propriété `PointInTimeRecoverySpecification` activée :

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Exemple de syntaxe de la requête :

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

Activation de la reprise ponctuelle (PITR) (API)

Exécutez l'opération d'API UpdateContinuousBackups avec le paramètre `PointInTimeRecoverySpecification` activé.

Exemple de syntaxe de la requête :

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

Exemple de syntaxe de la réponse :

```
{
   "ContinuousBackupsDescription": {
      "ContinuousBackupsStatus": "string",
      "PointInTimeRecoveryDescription": {
         "PointInTimeRecoveryStatus": "string",
         "EarliestRestorableDateTime": number,
         "RecoveryPeriodInDays": number,
         "LatestRestorableDateTime": number
      }
   }
}
```

Python

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.update_continuous_backups(
    TableName=<table_name>,
    PointInTimeRecoverySpecification={
        'PointInTimeRecoveryEnabled': True,
        'RecoveryPeriodInDays': 35
    }
)
```

Période de reprise

Vous pouvez définir la période de reprise pour les sauvegardes continues entre 1 et 35 jours. Cette valeur `RecoveryPeriodInDays` détermine la durée pendant laquelle vos sauvegardes continues sont maintenues. Par exemple, si vous définissez cette valeur sur 30 jours, vous ne pourrez restaurer votre table que de façon ponctuelle au cours des 30 derniers jours.

> **Note**
>
> DynamoDB facture la reprise ponctuelle (PITR) en fonction de la taille de chaque table DynamoDB, y compris les données de la table et les index secondaires locaux. La période de reprise maximale configurée n'a aucune incidence sur le prix qui vous est facturé pour l'activation de la PITR. Pour plus de détails sur la tarification, consultez DynamoDB pricing.

Modification de la reprise ponctuelle (PITR)

Vous pouvez modifier le paramètre PITR sur votre table et modifier la période de reprise. Si vous modifiez la période de reprise et que vous l'augmentez à une valeur supérieure à celle définie précédemment, votre valeur `EarliestRestorePoint` ne changera pas immédiatement. La période de reprise étant une période continue, DynamoDB continuera à effectuer des sauvegardes automatiques jusqu'à ce que la nouvelle période prolongée soit atteinte. Si vous modifiez la période de reprise et que vous la réduisez à une valeur inférieure à celle précédemment définie, votre valeur `EarliestRestorePoint` diminuera immédiatement pour correspondre à votre période de reprise, et toutes les sauvegardes continues qui ne respectent pas la nouvelle valeur définie ne seront pas récupérables.

Suppression d'une table avec l'activation de la reprise ponctuelle (PITR)

Lorsque vous supprimez une table et que la reprise ponctuelle (PITR) est activée, DynamoDB crée automatiquement un instantané de secours, appelé sauvegarde système, conservé pendant 35 jours (sans frais supplémentaires). Les sauvegardes du système vous permettent de restaurer la table supprimée à l'état qui était le sien avant la suppression. Toutes les sauvegardes du système suivent une convention de dénomination standard : *`table-name`*`$DeletedTableBackup`.

> **Note**
>
> Une fois qu'une table pour laquelle la reprise ponctuelle est activée est supprimée, vous pouvez utiliser la sauvegarde système pour restaurer cette table telle qu'elle était à un

moment précis. La sauvegarde système sera créée lors de la suppression de la table. Il s'agit d'un instantané de la table juste avant sa suppression.

Utilisation de la sauvegarde et de la restauration à la demande de DynamoDB

Amazon DynamoDB prend en charge les fonctionnalités de sauvegarde et de restauration à la demande autonomes. Ces fonctionnalités sont à votre disposition, que vous utilisiez AWS Backup ou non.

Vous pouvez utiliser la fonction de sauvegarde DynamoDB à la demande pour créer des sauvegardes complètes de vos tables pour l'archivage et la conservation à long terme, à des fins de conformité réglementaire. Vous pouvez sauvegarder et restaurer les données de votre table à tout moment d'un simple clic sur la console AWS de gestion ou d'un seul appel d'API. Les actions de sauvegarde et de restauration n'ont aucun impact sur les performances ou la disponibilité de la table.

Vous pouvez créer des sauvegardes de tables à l'aide de la console, de l'interface de ligne de AWS commande (AWS CLI) ou de l'API DynamoDB. Pour de plus amples informations, veuillez consulter Sauvegarde d'une table DynamoDB.

Pour en savoir plus sur la restauration d'une table à partir d'une sauvegarde, consultez Restauration d'une table DynamoDB à partir d'une sauvegarde.

Sauvegarde et restauration de tables DynamoDB avec DynamoDB : Fonctionnement

Vous pouvez utiliser la fonction de sauvegarde à la demande DynamoDB pour créer des sauvegardes complètes de vos tables Amazon DynamoDB. Cette fonction est disponible indépendamment de AWS Backup. Cette section présente le processus de sauvegarde et de restauration DynamoDB.

Sauvegardes

Lorsque vous créez une sauvegarde à la demande avec DynamoDB, un marqueur de temps est catalogué pour cette demande. La sauvegarde est créée de manière asynchrone par application de toutes les modifications jusqu'à l'heure de la demande sur l'instantané de la dernière table complète.

Les demandes de sauvegarde DynamoDB sont traitées instantanément et sont disponibles pour la restauration après quelques minutes.

> **Note**
>
> Chaque fois que vous créez une sauvegarde à la demande, toutes les données de la table sont sauvegardées. Il n'y a pas de limite au nombre de sauvegardes à la demande.

Toutes les sauvegardes dans DynamoDB fonctionnent sans consommer de débit approvisionné sur la table.

Les sauvegardes DynamoDB ne garantissent pas la cohérence causale entre les éléments. Toutefois, l'asymétrie entre les mises à jour d'une sauvegarde est généralement très inférieure à une seconde.

Lorsqu'une sauvegarde est en cours, vous ne pouvez pas effectuer les opérations suivantes :

- Suspendre ou annuler l'opération de sauvegarde.
- Supprimer la table source de la sauvegarde.
- Désactiver les sauvegardes d'une table si une sauvegarde de cette tables est en cours.

Si vous ne souhaitez pas créer des scripts de planification et des tâches de nettoyage, vous pouvez utiliser AWS Backup pour créer des plans de sauvegarde avec des planifications et des politiques de conservation pour vos tables DynamoDB. AWS Backup exécute les sauvegardes et les supprime quand elles expirent. Pour plus d'informations, consultez le Manuel du développeur AWS Backup.

En plus de AWS Backup, vous pouvez programmer des sauvegardes périodiques ou futures par l'intermédiaire des fonctions AWS Lambda. Pour plus d'informations, consultez le billet de blog A serverless solution to schedule your Amazon DynamoDB On-Demand backup.

Si vous utilisez la console, toutes les sauvegardes créées avec AWS Backup seront répertoriées sous l'onglet Backups (Sauvegardes), avec le Backup type (Type de sauvegarde) défini sur `AWS`.

> **Note**
>
> Vous ne pouvez pas supprimer des sauvegardes marquées avec un Backup type (Type de sauvegarde) AWS à l'aide de la console DynamoDB. Pour gérer ces sauvegardes, utilisez la console AWS Backup.

Pour savoir comment effectuer une sauvegarde, consultez Sauvegarde d'une table DynamoDB.

Restaurations

Vous restaurez une table sans consommer le débit alloué pour cette table. Vous pouvez effectuer une restauration complète de la table à partir de votre sauvegarde DynamoDB ou configurer les paramètres de la table de destination. Lorsque vous effectuez une restauration, vous pouvez modifier les paramètres de table suivants :

- Index secondaires globaux
- Index secondaires locaux
- Mode de facturation
- Capacité dimensionnée d'écriture et de lecture
- Paramètres de chiffrement

> **Important**
>
> Lorsque vous procédez à la restauration complète d'une table, la table de destination est définie avec les mêmes unités de capacité de lecture et d'écriture que la table source, telles qu'elles étaient enregistrées au moment de la demande de sauvegarde. Le processus de restauration restaure également les index secondaires locaux et les index secondaires globaux.

Vous pouvez également restaurer vos données de table DynamoDB entre régions AWS afin que la table restaurée soit créée dans une région différente de celle où réside la sauvegarde. Vous pouvez effectuer des restaurations entre régions commerciales AWS, régions AWS Chine et les régions AWS GovCloud (US). Vous payez uniquement pour les données que vous transférez hors de la région source et pour la restauration vers une nouvelle table dans la région de destination.

Les restaurations peuvent être plus rapides et plus économiques si vous choisissez de ne pas créer tout ou partie des index secondaires sur la nouvelle table restaurée.

Vous devez configurer manuellement les éléments suivants pour la table restaurée :

- Politiques de scalabilité automatique
- Gestion des identités et des accès AWS politiques (IAM)
- Métriques et alarmes Amazon CloudWatch
- Balises
- Paramètres de flux
- Paramètres de time-to-live (TTL)
- Paramètres de protection contre la suppression
- Paramètres de récupération ponctuelle (PITR)

Vous pouvez restaurer uniquement les données de l'ensemble d'une table dans une nouvelle table à partir d'une sauvegarde. Vous pouvez écrire sur la table restaurée seulement lorsque celle-ci est devenue active.

> **Note**
>
> Vous ne pouvez pas remplacer une table existante au cours d'une opération de restauration.

Les métriques de service montrent que 95 % des restaurations de table des clients prennent moins d'une heure. Toutefois, les temps de restauration sont directement liés à la configuration de vos tables (taille et nombre de partitions sous-jacentes), ainsi qu'à d'autres variables connexes. Une bonne pratique de planification de la reprise après sinistre consiste à documenter régulièrement les temps moyens de restauration et à déterminer comment ces délais affectent votre objectif global de temps de récupération.

Pour savoir comment effectuer une restauration, consultez Restauration d'une table DynamoDB à partir d'une sauvegarde.

Vous pouvez utiliser des politiques IAM pour le contrôle d'accès. Pour de plus amples informations, consultez Utilisation d'IAM avec la sauvegarde et la restauration dans DynamoDB.

Toutes les actions de sauvegarde et de restauration de la console et de l'API sont capturées et enregistrées dans AWS CloudTrail pour la journalisation, la surveillance en continu et l'audit.

Sauvegarde d’une table DynamoDB

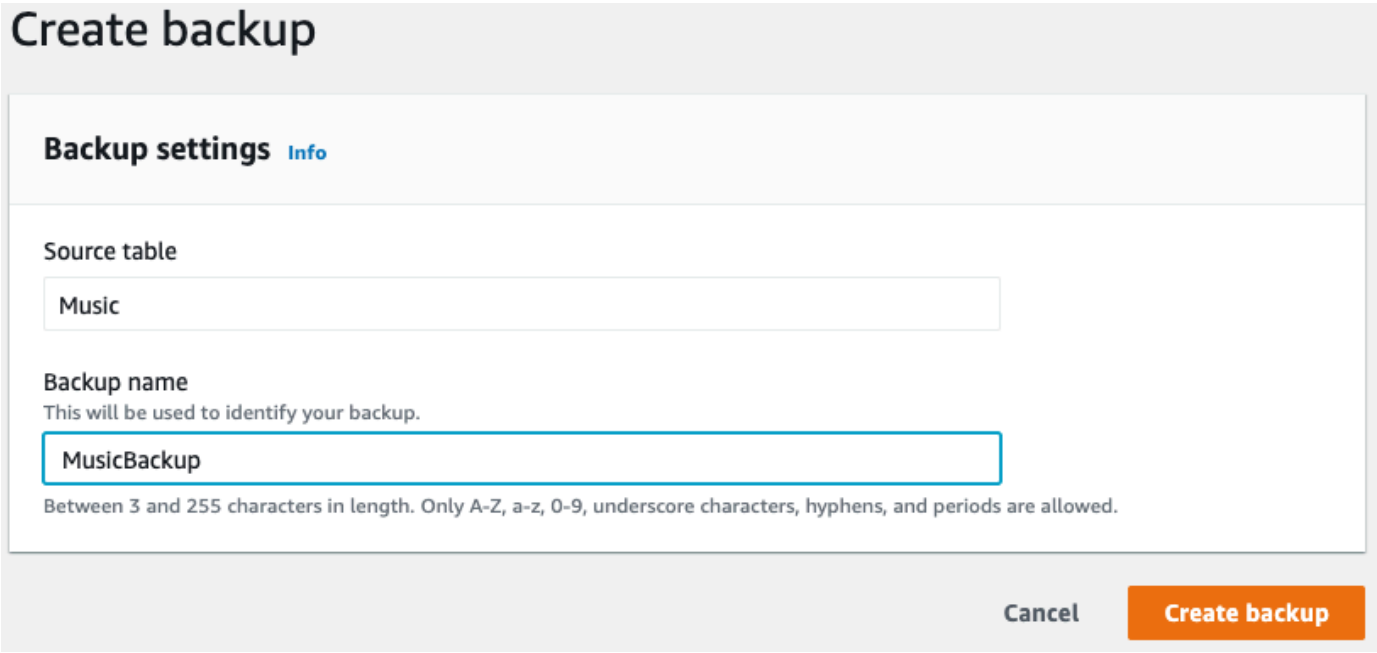
Cette section décrit comment utiliser la console Amazon DynamoDB ou AWS Command Line Interface comment sauvegarder une table.

Création d’une sauvegarde de table (console)

Suivez ces étapes pour créer une sauvegarde nommée `MusicBackup` pour une table `Music` existante à l’aide de la AWS Management Console.

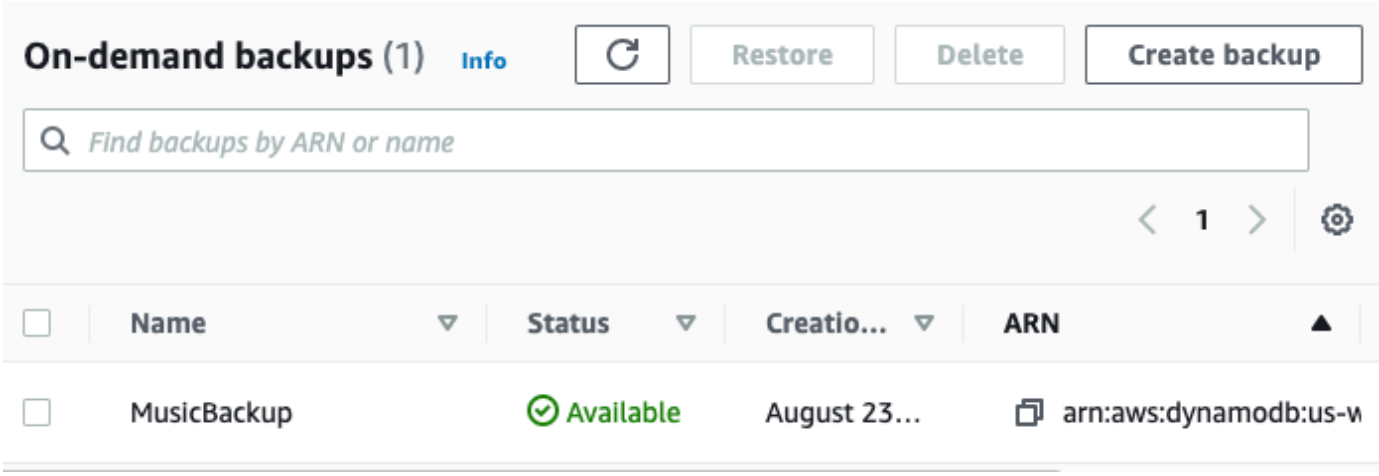
Pour créer une sauvegarde de table

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Vous pouvez créer une sauvegarde en procédant de l’une des manières suivantes :
   - Dans l’onglet Sauvegardes de la table `Music`, choisissez Créer une sauvegarde.
   - Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes. Choisissez ensuite Créer une sauvegarde.
3. Assurez-vous qu’il s’agit bien du nom de la table `Music` et saisissez **`MusicBackup`** comme nom de sauvegarde. Choisissez ensuite Créer une sauvegarde pour créer la sauvegarde.



> **Note**
>
> Si vous créez des sauvegardes par l'intermédiaire de la section Sauvegardes du volet de navigation, la table n'est pas présélectionnée pour vous. Vous devez choisir manuellement le nom de la table source pour la sauvegarde.

Pendant la création de la sauvegarde, le statut de cette dernière est Création. Lorsque la sauvegarde est terminée, l'état devient Disponible.



Création d'une sauvegarde de table (AWS CLI)

Suivez ces étapes pour créer une sauvegarde pour une table `Music` existante à l'aide de l'AWS CLI.

Pour créer une sauvegarde de table

- Créez une sauvegarde nommée `MusicBackup` pour la table `Music`.

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

Pendant la création de la sauvegarde, l'état de cette dernière est `CREATING`.

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
```

```
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Une fois la sauvegarde terminée, l'état `BackupStatus` doit passer à `AVAILABLE`. Pour confirmer, utilisez la commande `describe-backup`. Vous pouvez obtenir la valeur d'entrée de `backup-arn` à partir de la sortie de l'étape précédente ou par l'intermédiaire de la commande `list-backups`.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Pour conserver une trace de vos sauvegardes, vous pouvez utiliser la commande `list-backups`. Elle permet d'afficher la liste de vos sauvegardes dont l'état est `CREATING` ou `AVAILABLE`.

```
aws dynamodb list-backups
```

Les commandes `list-backups` et `describe-backup` sont utiles pour vérifier les informations relatives à la table source de la sauvegarde.

Restauration d'une table DynamoDB à partir d'une sauvegarde

Cette section explique comment restaurer une table à partir d'une sauvegarde à l'aide de la console Amazon DynamoDB ou AWS Command Line Interface du ().AWS CLI

> **Note**
>
> Si vous souhaitez utiliser le AWS CLI, vous devez d'abord le configurer. Pour de plus amples informations, veuillez consulter Accès à DynamoDB.

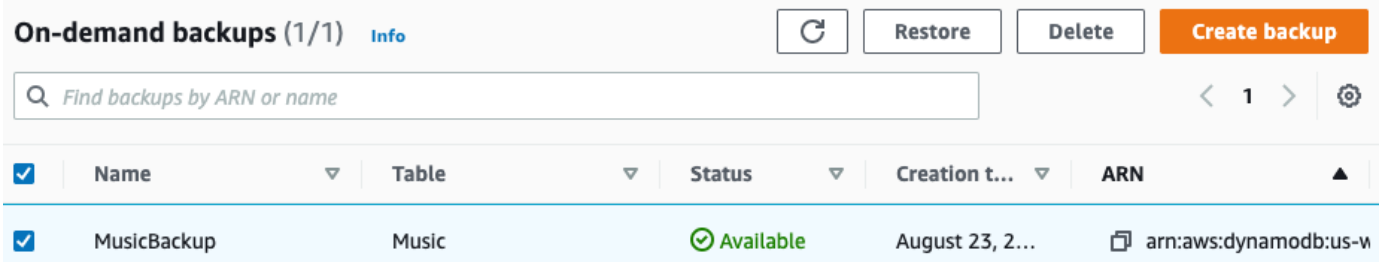
Restauration d'une table à partir d'une sauvegarde (console)

La procédure suivante décrit comment restaurer la table `Music` à l'aide du fichier `MusicBackup` créé dans le didacticiel Sauvegarde d'une table DynamoDB.

> **Note**
>
> Cette procédure suppose que la table `Music` n'existe plus avant de la restaurer à l'aide du fichier `MusicBackup`.

Pour restaurer une table à partir d'une sauvegarde

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Dans la liste des sauvegardes, choisissez `MusicBackup`.



4. Choisissez Restore (Restaurer).
5. Entrez **`Music`** comme nom de la nouvelle table. Confirmez le nom de la sauvegarde et les autres détails de la sauvegarde. Choisissez ensuite Restore table pour démarrer le processus de restauration.

> **Note**
>
> Vous pouvez restaurer la table dans la même AWS région ou dans une région différente de celle où réside la sauvegarde. Vous pouvez également exclure la création d'index secondaires sur la nouvelle table restaurée. En outre, vous pouvez spécifier un mode de chiffrement différent.
>
> Les tables restaurées à partir de sauvegardes sont toujours créées à l'aide de la classe de tables DynamoDB Standard.

Restore table from backup Info

Restoring a table from a backup will restore it as a new table.

Restore settings

Name of restored table

This name will identify your restored table.

Music

Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.

Secondary indexes

- (•) Restore the entire table
  Your restored table will include all local and global secondary indexes.
- ( ) Restore the table without secondary indexes
  Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.

Destination AWS Region

- (•) Same Region (Oregon)
  Restore the table to the same Region as the original table.
- ( ) Cross-Region
  Restore the table to a different Region for greater redundancy but with higher data transfer costs.

▼ Encryption at rest - *optional*

All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.

Encryption key management Info

- (•) Owned by Amazon DynamoDB
  The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).
- ( ) AWS managed CMK
  The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.
- ( ) Stored in your account, and owned and managed by you
  Choose a key that is owned and managed by you, and stored in AWS KMS.

ⓘ The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more

Cancel Restore

La table en cours de restauration s'affiche avec le statut Creating. Une fois le processus de restauration terminé, l'état de la table `Music` devient Active.

Restauration d'une table à partir d'une sauvegarde (AWS CLI)

Suivez ces étapes pour restaurer la `Music` table AWS CLIà l'aide de `MusicBackup` celle créée dans le Sauvegarde d'une table DynamoDB didacticiel.

Pour restaurer une table à partir d'une sauvegarde

1. Confirmez la sauvegarde à restaurer à l'aide de la commande `list-backups`. Cet exemple utilise `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Pour obtenir des détails supplémentaires pour la sauvegarde, utilisez la commande `describe-backup`. Vous pouvez accéder à l'entrée `backup-arn` issue de l'étape précédente.

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Restaurez la table à partir de la sauvegarde. Dans ce cas, la `MusicBackup` `Music` table est restaurée dans la même AWS région.

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. Restaurez la table à partir de la sauvegarde avec des paramètres de table personnalisés. Dans ce cas, `MusicBackup` restaure la table `Music` et spécifie un mode de chiffrement pour la table restaurée.

   > **Note**
   >
   > Le paramètre `sse-specification-override` prend les mêmes valeurs que le paramètre `sse-specification-override` utilisé dans la commande `CreateTable`.

Pour en savoir plus, veuillez consulter la section Gestion des tables chiffrées dans DynamoDB.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Vous pouvez restaurer la table dans une AWS région différente de celle où réside la sauvegarde.

Note

- Le paramètre `sse-specification-override` est obligatoire pour les restaurations entre régions, mais facultatif pour les restaurations dans la même région que celle de la table source.
- Lorsque vous effectuez une restauration entre régions à partir de la ligne de commande, vous devez définir la AWS région par défaut sur la région de destination souhaitée. Pour en savoir plus, consultez Options de ligne de commande dans le Guide de l'utilisateur AWS Command Line Interface.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Vous pouvez remplacer le mode de facturation et le débit alloué pour la table restaurée.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
--billing-mode-override PAY_PER_REQUEST
```

Vous pouvez exclure la création d'index secondaires sur la table restaurée.

> **Note**
>
> Les restaurations peuvent être plus rapides et plus économiques si vous empêchez la création de certains index ou de tous les index secondaires sur la table restaurée.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

> **Note**
>
> Les index secondaires fournis doivent correspondre aux index existants. Vous ne pouvez pas créer de nouveaux index au moment de la restauration.

Vous pouvez utiliser une combinaison de remplacements différents. Par exemple, vous pouvez utiliser un index secondaire global unique et modifier le débit alloué en même temps, comme suit.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Afin de vérifier la restauration, utilisez la commande `describe-table` pour décrire la table `Music`.

```
aws dynamodb describe-table --table-name Music
```

La table en cours de restauration à partir de la sauvegarde s'affiche avec le statut Creating. Une fois le processus de restauration terminé, l'état de la table `Music` devient Active.

> **Important**
>
> Ne modifiez ou ne supprimez pas votre politique de rôle IAM quand une restauration est en cours, car cela pourrait entraîner un comportement inattendu. Par exemple, supposons que vous ayez supprimé les autorisations d'écriture pour une table en cours de restauration. Dans ce cas, l'opération `RestoreTableFromBackup` sous-jacente ne pourra pas écrire les données restaurées dans la table.
> Une fois l'opération de restauration terminée, vous pouvez modifier ou supprimer votre politique de rôle IAM.
> Les politiques IAM impliquant des restrictions d'adresses IP sources pour accéder à la table de restauration cible doivent avoir la clé `aws:ViaAWSService` définie sur `false` pour s'assurer que les restrictions s'appliquent uniquement aux demandes faites directement par un principal. Sinon, la restauration est annulée.
> Si votre sauvegarde est chiffrée à l'aide d'une clé gérée par le client Clé gérée par AWS ou d'une clé gérée par le client, ne désactivez pas ou ne supprimez pas la clé pendant qu'une restauration est en cours, sinon la restauration échouera.
> Une fois l'opération de restauration terminée, vous pouvez modifier la clé de chiffrement de la table restaurée et désactiver ou supprimer l'ancienne clé.

Suppression d'une sauvegarde de table DynamoDB

Cette section décrit comment utiliser le AWS Management Console ou le AWS Command Line Interface(AWS CLI) pour supprimer une sauvegarde de table Amazon DynamoDB.

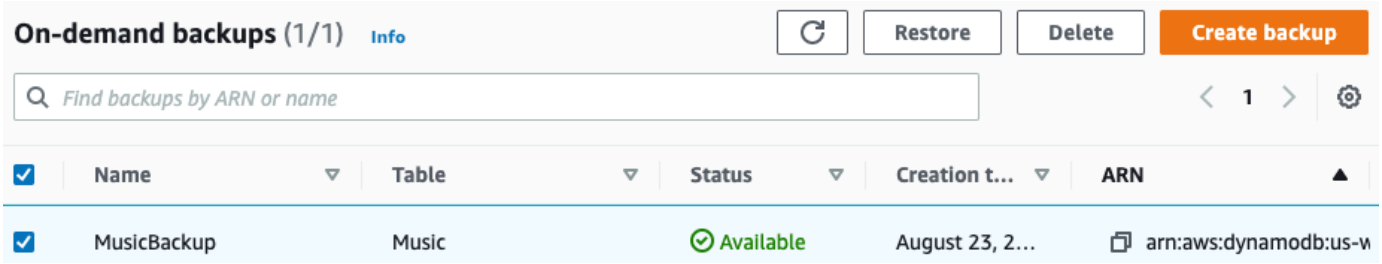
> **Note**
>
> Si vous souhaitez utiliser le AWS CLI, vous devez d'abord le configurer. Pour de plus amples informations, veuillez consulter À l'aide du AWS CLI.

Suppression d'une sauvegarde de table (console)

La procédure suivante décrit comment utiliser la console pour supprimer le fichier `MusicBackup` créé dans le didacticiel Sauvegarde d'une table DynamoDB.

Pour supprimer une sauvegarde

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Dans la liste des sauvegardes, choisissez `MusicBackup`.



4. Sélectionnez Delete (Supprimer). Confirmer que vous souhaitez supprimer la sauvegarde en saisissant **`delete`** et en cliquant sur Supprimer.

Suppression d'une sauvegarde de table (AWS CLI)

L'exemple suivant supprime une sauvegarde pour une table `Music` existante à l'aide de l'AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Utilisation d'IAM avec la sauvegarde et la restauration dans DynamoDB

Vous pouvez utiliser Gestion des identités et des accès AWS(IAM) pour restreindre les actions de sauvegarde et de restauration Amazon DynamoDB pour certaines ressources. Les `CreateBackup` et `RestoreTableFromBackup` APIs fonctionnent sur une base par table.

Pour plus d'informations sur l'utilisation de politiques IAM dans DynamoDB, consultez Politiques basées sur l'identité pour DynamoDB.

Vous trouverez ci-dessous des exemples de politiques IAM que vous pouvez utiliser pour configurer une fonctionnalité de sauvegarde et restauration spécifique dans DynamoDB.

Exemple 1 : Autoriser les RestoreTableFromBackup actions CreateBackup et

La politique IAM suivante autorise les actions DynamoDB `CreateBackup` et `RestoreTableFromBackup` sur toutes les tables :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

Important

Les autorisations RestoreTableFromBackup DynamoDB sont nécessaires sur la sauvegarde source, et les autorisations de lecture et d'écriture DynamoDB sur la table cible sont nécessaires pour la fonctionnalité de restauration.
Les autorisations RestoreTableToPointInTime DynamoDB sont nécessaires sur la table source, et les autorisations de lecture et d'écriture DynamoDB sur la table cible sont nécessaires pour la fonctionnalité de restauration.

Exemple 2 : autoriser CreateBackup et refuser RestoreTableFromBackup

La politique IAM suivante autorise l'action `CreateBackup` et rejette l'action `RestoreTableFromBackup` :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Exemple 3 : autoriser ListBackups et refuser CreateBackup et RestoreTableFromBackup

La politique IAM suivante autorise l'action `ListBackups` et rejette les actions `CreateBackup` et `RestoreTableFromBackup` :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Exemple 4 : autoriser ListBackups et refuser DeleteBackup

La politique IAM suivante autorise l'action `ListBackups` et rejette l'action `DeleteBackup` :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

Exemple 5 : Autoriser RestoreTableFromBackup et DescribeBackup pour toutes les ressources et refuser DeleteBackup pour une sauvegarde spécifique

La politique IAM suivante autorise les actions `RestoreTableFromBackup` et `DescribeBackup`, et rejette l'action `DeleteBackup` pour une ressource de sauvegarde spécifique :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
}
```

> **Important**
>
> Les autorisations RestoreTableFromBackup DynamoDB sont nécessaires sur la sauvegarde source, et les autorisations de lecture et d'écriture DynamoDB sur la table cible sont nécessaires pour la fonctionnalité de restauration.
> Les autorisations RestoreTableToPointInTime DynamoDB sont nécessaires sur la table source, et les autorisations de lecture et d'écriture DynamoDB sur la table cible sont nécessaires pour la fonctionnalité de restauration.

Exemple 6 : CreateBackup Autoriser un tableau spécifique

La politique IAM suivante autorise l'action `CreateBackup` sur la table `Movies` uniquement :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

Exemple 7 : Autoriser ListBackups

La politique IAM suivante autorise l'action `ListBackups` :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> Vous ne pouvez pas autoriser l'action `ListBackups` sur une table spécifique.

Exemple 8 : Autoriser l'accès aux AWS Backup fonctionnalités

Vous aurez besoin des autorisations API pour l'action `StartAwsBackupJob` en vue d'une sauvegarde réussie avec des fonctions avancées, et pour l'action `dynamodb:RestoreTableFromAwsBackup` en vue d'une restauration réussie de cette sauvegarde.

La politique IAM suivante accorde AWS Backup les autorisations nécessaires pour déclencher des sauvegardes avec des fonctionnalités avancées et des restaurations. Notez également que si les tables sont chiffrées, la politique devra accéder à la clé KMS AWS.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
```

```
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

Exemple 9 : Refus RestoreTableToPointInTime pour une table source spécifique

La stratégie IAM suivante refuse les autorisations pour l'action `RestoreTableToPointInTime` pour une table source spécifique :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Exemple 10 : Refus RestoreTableFromBackup de toutes les sauvegardes pour une table source spécifique

La stratégie IAM suivante refuse les autorisations pour l'action `RestoreTableToPointInTime` pour toutes les sauvegardes d'une table source spécifique :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

Présentation de la facturation des sauvegardes pour Amazon DynamoDB

Ce guide fournit des informations détaillées sur le fonctionnement de la facturation des sauvegardes DynamoDB. Nous analyserons les différents éléments qui contribuent au coût global, en fournissant des explications claires et des exemples pratiques.

DynamoDB propose des sauvegardes à la demande et de reprise ponctuelle (PITR) pour protéger vos données DynamoDB en cas de sinistre. Il propose également un archivage des données pour une conservation à long terme.

Comment ça marche

Les sauvegardes à la demande DynamoDB sont facturées mensuellement. Si vous effectuez une sauvegarde un jour donné du mois, vous verrez une facturation unique pour cette sauvegarde, calculée pour les jours restants du mois (par exemple, si vous créez une sauvegarde le 27, vous ne serez facturé que pour les quelques jours restants du mois, avec une facturation unique le 27).

Si vous conservez vos sauvegardes précédentes pour les mois suivants, un mois complet sera facturé pour cette sauvegarde, le 1er du mois. Si la sauvegarde est supprimée avant la fin du mois, les frais seront ajustés en fonction de l’utilisation réelle.

Par exemple, si vous avez créé une sauvegarde le 27 juillet et que vous la conservez pendant le mois d'août, les frais suivants s'appliqueront à cette sauvegarde :

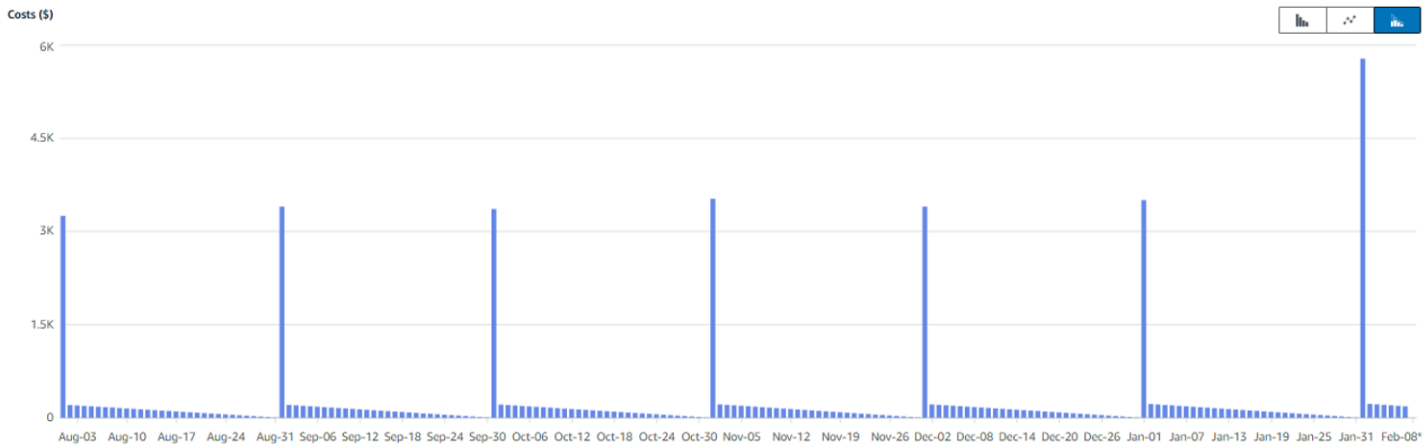
- des frais le 27 juillet pour les jours de juillet restants ;
- des frais le 1er août pour tout le mois d'août ;
- des frais le 1er de chaque mois suivant pendant lequel la sauvegarde existe.

Lorsque des sauvegardes sont conservées pour les tables DynamoDB, vous pouvez observer que les dépenses pour la métrique d'utilisation `DynamoDB (Region)-TimedBackupStorage-ByteHrs` semblent anormalement élevées le 1er du mois. En outre, si vous vérifiez cette métrique au début d'un nouveau mois et que vous la comparez aux cycles de facturation précédents, vous pouvez observer ce qui semble être un pic d'utilisation important. Ce comportement est intégré à la conception. Le 1er de chaque mois, toutes les sauvegardes DynamoDB existantes sont soumises à des frais d'utilisation pour l'ensemble du mois. Les frais d'utilisation des sauvegardes DynamoDB supprimées au cours du mois seront calculés au prorata de l'utilisation réelle. Par conséquent, il est possible que les frais (appliqués le 1er du mois) diminuent au cours du mois. En effet, les politiques de conservation appliquent des expirations ou des suppressions manuelles aux sauvegardes reportées. Ce point sera examiné dans un scénario ci-dessous.

De même, vous remarquerez de faibles pics tout au long du mois à mesure que de nouvelles sauvegardes seront créées, les frais étant appliqués le jour de leur création pour le reste du mois.

Exemple de facturation de sauvegarde DynamoDB

Voici un exemple de ce que vous pouvez voir dans Cost Explorer au début du mois :



Vous remarquerez que le 1er février semble connaître un pic beaucoup plus important que les mois précédents. Découvrons pourquoi.

Citation tirée de la page de tarification de DynamoDB :

« La taille totale du stockage de sauvegarde facturée chaque mois est la somme de toutes les sauvegardes des tables DynamoDB. DynamoDB surveille la taille des sauvegardes à la demande en continu tout au long du mois afin de déterminer vos frais de sauvegarde. »

Cela explique pourquoi la facture présente systématiquement un pic d'utilisation important le 1er de chaque mois. Toutes les sauvegardes existantes arrivant au cours d'un nouveau mois sont facturées pour un mois complet à compter du 1er. Autrement dit, si vous commencez le mois avec 300 sauvegardes DynamoDB, les frais d'utilisation d'un mois complet sont appliqués le 1er jour du mois pour l'ensemble des 300 sauvegardes.

Les nouvelles sauvegardes effectuées tout au long du mois entraîneront des frais d'utilisation à compter du jour de leur création jusqu'à la fin du mois.

Et si la sauvegarde est supprimée au milieu du mois ?

Voici quelques scénarios à envisager :

1. Si une sauvegarde du mois précédent est supprimée le 15 du mois en cours, les frais d'utilisation de cette sauvegarde, toujours appliqués le 1er, seront ajustés en fonction de l'utilisation réelle au lieu du mois complet supposé d'utilisation précédemment appliqué. L'exemple ci-dessous explique cela plus en détail.
2. Lorsque vous créez une sauvegarde au cours du mois, les frais d'utilisation pour le reste du mois sont appliqués au jour de sa création. Cependant, si vous supprimez cette sauvegarde avant la fin du mois, vos frais d'utilisation seront ajustés au prorata du nombre de jours pendant lesquels la sauvegarde est restée active, tout en étant appliqués à la date de création initiale.

Pourquoi l'utilisation du mois en cours semble-t-elle beaucoup plus élevée le 1er du mois que les mois précédents, et que se passe-t-il si je supprime les sauvegardes ?

Pour répondre à cette importante question en deux parties, établissons un scénario à l'aide des informations suivantes :

- Durée du mois : 30 jours
- Fréquence de sauvegarde DynamoDB : 10 par jour, 300 par mois

- Politique de conservation des sauvegardes DynamoDB : 30 jours
- Coût DynamoDB par sauvegarde : 2 USD par jour, 60 USD par mois
- Total au 1er du mois précédent (`TimedBackupStorage-ByteHrs`, vérifié le 1er du mois en cours) : 9 300 USD
- Total du mois précédent (`TimedBackupStorage-ByteHrs`) : 18 600 USD
- Total au 1er du mois en cours (`TimedBackupStorage-ByteHrs`, vérifié le 1er du mois en cours) : 18 000 USD
- Modifications de l'utilisation de DynamoDB d'un mois à l'autre : aucune

À l'aide des informations ci-dessus, nous pouvons constater que 300 sauvegardes ont été créées le mois précédent avec une politique de conservation de 30 jours. Le 1er d'un nouveau mois, toutes ces sauvegardes sont encore présentes, car elle n'ont pas encore atteint la fin de leur période de conservation. Cependant, au fil des jours, les ensembles de sauvegardes les plus anciens commenceront à expirer et à être supprimés, comme illustré ici :

Tableau de décompte des sauvegardes DynamoDB

Nouveau mois	Jour 1	Jour 2	Jour 3	Jour 4	Jour 5
Nombre total de sauvegardes du mois précédent reportées	300	290	280	270	260

- Le 1er, nous constatons 300 sauvegardes à 60 USD par mois par sauvegarde, soit un total de 18 000 USD de `TimedBackupStorage-ByteHrs` appliqués. Cela contraste avec le mois précédent, où le total du mois était de 18 600 USD.
- Le 2, 10 de ces sauvegardes auront expiré et ne seront plus facturées. Dans ce cas, les frais appliqués à ces sauvegardes seront ajustés en fonction de l'utilisation réelle plutôt que de l'utilisation présumée. Ces 10 sauvegardes, qui étaient auparavant facturées 600 USD le 1er (10 sauvegardes x 30 jours), sont donc ramenées à 20 USD (10 sauvegardes x 1 jour).
- Le jour suivant, le bloc suivant de 10 sauvegardes expirera et sera supprimé, réduisant leur utilisation de 30 jours à 2 jours, abaissant leur coût à 40 USD (10 sauvegardes x 2 jours).

Au fil des jours, on verra ce pic – plus important que celui du mois précédent – diminuer progressivement. Si nous étendons cette vue à l'ensemble du mois, nous observerons ce qui suit :

Évolution des frais de sauvegarde DynamoDB (1er du mois)

300 sauvegardes par blocs de 10	1er	10 du mois	20 du mois	30 du mois
Bloc 1	600 USD	20 USD	20 USD	20 USD
Bloc 2	600 USD	40 USD	40 USD	40 USD
Bloc 3	600 USD	60 USD	60 USD	60 USD
Bloc 4	600 USD	80 USD	80 USD	80 USD
Bloc 5	600 USD	100 USD	100 USD	100 USD
Bloc 6	600 USD	120 USD	120 USD	120 USD
Bloc 7	600 USD	140 USD	140 USD	140 USD
Bloc 8	600 USD	160 USD	160 USD	160 USD
Bloc 9	600 USD	180 USD	180 USD	180 USD
Bloc 10	600 USD	600 USD	USD200	USD200
Bloc 11	600 USD	600 USD	220 USD	220 USD
Bloc 12	600 USD	600 USD	240 USD	240 USD
Bloc 13	600 USD	600 USD	260 USD	260 USD
Bloc 14	600 USD	600 USD	280 USD	280 USD
Bloc 15	600 USD	600 USD	USD300	USD300
Bloc 16	600 USD	600 USD	320 USD	320 USD
Bloc 17	600 USD	600 USD	340 USD	340 USD

300 sauvegardes par blocs de 10	1er	10 du mois	20 du mois	30 du mois
Bloc 18	600 USD	600 USD	360 USD	360 USD
Bloc 19	600 USD	600 USD	380 USD	380 USD
Bloc 20	600 USD	600 USD	600 USD	400 USD
Bloc 21	600 USD	600 USD	600 USD	420 USD
Bloc 22	600 USD	600 USD	600 USD	440 USD
Bloc 23	600 USD	600 USD	600 USD	460 USD
Bloc 24	600 USD	600 USD	600 USD	480 USD
Bloc 25	600 USD	600 USD	600 USD	500 USD
Bloc 26	600 USD	600 USD	600 USD	520 USD
Bloc 27	600 USD	600 USD	600 USD	540 USD
Bloc 28	600 USD	600 USD	600 USD	560 USD
Bloc 29	600 USD	600 USD	600 USD	580 USD
Bloc 30	600 USD	600 USD	600 USD	600 USD
Total au 1er du mois (USD)	18 000 USD	13 500 USD	10 400 USD	9 300 USD

À mesure qu'un nouveau bloc de sauvegardes disparaît chaque jour, son utilisation est ajustée en fonction du nombre de jours pendant lesquels il a été actif, plutôt que sur le montant total du mois. En conséquence, à la fin du mois, les frais initialement observés le 1er (18 000 USD) auront diminué pour atteindre les 9 300 USD prévus. Ce montant, combiné aux nouvelles sauvegardes créées au cours du mois (qui auront un tableau de facturation similaire à celui ci-dessus, mais en sens inverse), aboutira à une dépense mensuelle totale proche des 18 600 USD.

Restauration d'une table dans DynamoDB

Vous pouvez restaurer une table DynamoDB à partir de votre sauvegarde PITR ou de vos sauvegardes à la demande à l'aide de la AWS Management Console, de l'interface de ligne de commande d'AWS (AWS CLI) ou de l'API DynamoDB. Le processus de reprise restaure une nouvelle table DynamoDB.

Restauration d'une table à l'aide de la reprise ponctuelle

Vous pouvez restaurer votre table à n'importe quel moment dans le passé, jusqu'au `EarliestRestoreableDateTime`.

> **Important**
>
> Si vous désactivez la reprise ponctuelle pour l'activer ultérieurement sur une table, vous réinitialisez l'heure de début à laquelle vous pouvez restaurer cette table. Ainsi, vous pouvez uniquement restaurer instantanément cette table à l'aide de `LatestRestorableDateTime`.

Lorsque vous restaurez à l'aide de la reprise ponctuelle, DynamoDB restaure vos données de table en fonction de la date et de l'heure sélectionnées (day:hour:minute:second) dans une nouvelle table. Vous restaurez une table sans utiliser le débit provisionné pour cette table. Vous pouvez effectuer une restauration complète de la table à l'aide de la restauration à un instant dans le passé ou configurer les paramètres de la table de destination. Vous pouvez modifier les paramètres de table suivants sur la table restaurée :

- Index secondaires globaux
- Index secondaires locaux
- Mode de facturation
- Capacité dimensionnée d'écriture et de lecture
- Paramètres de chiffrement

> **Important**
>
> Lorsque vous procédez à la restauration complète d'une table, la table de destination est définie avec les mêmes unités de capacité de lecture et d'écriture provisionnées que la table

source avait au moment de la demande de sauvegarde. Supposons par exemple que le débit alloué d'une table vienne d'être abaissé à 50 unités de capacité de lecture et 50 unités de capacité d'écriture. Vous restaurez ensuite la table à l'état qui était le sien trois semaines auparavant, alors que son débit alloué était défini sur 100 unités de capacité en lecture et 100 unités de capacité en écriture. Dans ce cas, DynamoDB restaure les données de votre table à cet instant dans le passé avec le débit alloué (100 unités de capacité de lecture et 100 unités de capacité d'écriture).

Vous pouvez également restaurer les données de votre table DynamoDB dans toutes les Régions AWS afin que la table restaurée soit créée dans une région différente de celle où réside la table source. Vous pouvez effectuer des restaurations entre régions commerciales AWS, régions AWS Chine et AWS GovCloud (US). Vous payez uniquement pour les données que vous transférez hors de la région source et pour la restauration vers une nouvelle table dans la région de destination.

> **Note**
>
> La restauration entre régions n'est pas prise en charge si la région source ou de destination est la région Asie-Pacifique (Hong Kong) ou la région Moyen-Orient (Bahreïn).

Les restaurations peuvent être plus rapides et plus économiques si vous empêchez la création de certains index ou de tous les index sur la table restaurée. Vous devez configurer manuellement les éléments suivants pour la table restaurée :

- Politiques d'autoscaling
- Politiques Gestion des identités et des accès AWS
- Métriques et alarmes Amazon CloudWatch Events
- Balises
- Paramètres de flux
- Paramètres de time-to-live (TTL)
- Paramètres de la restauration à un instant dans le passé

Le temps nécessaire pour restaurer une table varie en fonction de plusieurs facteurs et n'est pas toujours corrélé à la taille de la table.

Restauration d'une table DynamoDB à un instant dans le passé

La restauration à un instant dans le passé d'Amazon DynamoDB fournit des sauvegardes continues des données de votre table DynamoDB. Vous pouvez restaurer une table à un instant dans le passé à l'aide de la console AWS Command Line Interface ou de l'AWS CLI (). Le processus de restauration à un instant dans le passé restaure une nouvelle table.

Si vous souhaitez utiliser l'AWS CLI, vous devez d'abord la configurer. Pour plus d'informations, consultez Accès à DynamoDB.

Rubriques

- Restauration d'une table DynamoDB à un instant dans le passé (console)
- Restauration d'une table à un instant dans le passé (AWS CLI)

Restauration d'une table DynamoDB à un instant dans le passé (console)

L'exemple suivant montre comment utiliser la console DynamoDB pour restaurer une table existante nommée `Music` à un instant dans le passé.

> **Note**
>
> Cette procédure suppose que vous ayez activé la restauration à un instant dans le passé. Pour l'activer pour la table `Music`, dans l'onglet Sauvegardes, dans la section Reprise ponctuelle (PITR), choisissez Modifier puis cochez la case à côté de Activer la restauration à un instant dans le passé.

Pour restaurer une table à un instant dans le passé

1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Dans la liste des tables, choisissez la table `Music`.
4. Dans l'onglet Sauvegardes de la table `Music`, dans la section Récupération à un instant donné dans le passé (PITR), choisissez Restaurer.
5. Pour le nom de la nouvelle table, tapez **`MusicMinutesAgo`**.

> **Note**
>
> Vous pouvez restaurer la table dans la même région AWS que la table source ou dans une région différente. Vous pouvez également exclure la création d'index secondaires sur la table restaurée. En outre, vous pouvez spécifier un mode de chiffrement différent.

6. Pour confirmer l'heure de restauration, définissez la date et l'heure de restauration sur Plus tôt. Choisissez ensuite Restaurer pour démarrer le processus de restauration.

   La table en cours de restauration s'affiche avec le statut Restauration en cours. Une fois le processus de restauration terminé, l'état de la table `MusicMinutesAgo` devient Active.

Restauration d'une table à un instant dans le passé (AWS CLI)

La procédure suivante montre comment utiliser l'AWS CLI pour restaurer une table existante nommée `Music` à un instant dans le passé.

> **Note**
>
> Cette procédure suppose que vous ayez activé la restauration à un instant dans le passé. Pour l'activer pour la table `Music`, exécutez la commande suivante.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Pour restaurer une table à un instant dans le passé

1. Confirmez l'activation de la restauration à un instant dans le passé pour la table `Music` à l'aide de la commande `describe-continuous-backups`.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   Les sauvegardes continues (automatiquement activées lors de la création de la table) et la restauration à un instant dans le passé sont activées.

```
{
    "ContinuousBackupsDescription": {
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": 1519257118.0,
            "LatestRestorableDateTime": 1520018653.01
        },
        "ContinuousBackupsStatus": "ENABLED"
    }
}
```

2. Restaurez la table à un instant dans le passé. Dans ce cas, la table `Music` est restaurée à `LatestRestorableDateTime` (il y a environ 5 minutes) dans la même région AWS.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

> **Note**
>
> Vous pouvez également restaurer à un instant spécifique dans le passé. Pour ce faire, exécutez la commande à l'aide de l'argument `--restore-date-time` et spécifiez un horodatage. Vous pouvez spécifier n'importe quel instant spécifique au cours de la période de restauration configurée, qui peut être définie sur une valeur comprise entre 1 et 35 jours. Par exemple, la commande suivante restaure la table à la `EarliestRestorableDateTime`.
>
> ```
> aws dynamodb restore-table-to-point-in-time \
>     --source-table-name Music \
>     --target-table-name MusicEarliestRestorableDateTime \
>     --no-use-latest-restorable-time \
>     --restore-date-time 1519257118.0
> ```
>
> La spécification de l'argument `--no-use-latest-restorable-time` est facultative lors de la restauration à un instant spécifique dans le passé.

3. Restaurez la table à un instant dans le passé avec des paramètres de table personnalisés. Dans ce cas, la table `Music` est restaurée à la `LatestRestorableDateTime` (il y a environ 5 minutes).

   Vous pouvez spécifier un mode de chiffrement différent pour la table restaurée, comme suit.

   > **Note**
   >
   > Le paramètre `sse-specification-override` prend les mêmes valeurs que le paramètre `sse-specification-override` utilisé dans la commande `CreateTable`. Pour en savoir plus, veuillez consulter la section Gestion des tables chiffrées dans DynamoDB.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time \
       --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
   abcd-1234-a123-ab1234a1b234
   ```

   Vous pouvez restaurer la table dans une région AWS différente de celle où réside la table source.

   > **Note**
   >
   > - Le paramètre `sse-specification-override` est obligatoire pour les restaurations entre régions, mais facultatif pour les restaurations dans la même région que celle de la table source.
   > - Le paramètre `source-table-arn` doit être fourni pour les restaurations entre régions.
   > - Lorsque vous effectuez une restauration entre régions à partir de la ligne de commande, vous devez définir la région AWS par défaut sur la région de destination souhaitée. Pour en savoir plus, consultez Options de ligne de commande dans le Guide de l'utilisateur AWS Command Line Interface.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Vous pouvez remplacer le mode de facturation et le débit alloué pour la table restaurée.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

Vous pouvez exclure la création d'index secondaires sur la table restaurée.

> **Note**
>
> Les restaurations peuvent être plus rapides et plus économiques si vous empêchez la création de certains index ou de tous les index secondaires sur la nouvelle table restaurée.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

Vous pouvez utiliser une combinaison de remplacements différents. Par exemple, vous pouvez utiliser un index secondaire global unique et modifier le débit alloué en même temps, comme suit.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
```

```
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

Afin de vérifier la restauration, utilisez la commande `describe-table` pour décrire la table `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

La table en cours de restauration est affichée avec le statut Création en cours et la restauration en cours avec la valeur vrai. Une fois le processus de restauration terminé, l'état de la table `MusicEarliestRestorableDateTime` devient Active.

> **Important**
>
> Pendant la restauration, ne modifiez ou ne supprimez pas les politiques Gestion des identités et des accès AWS (IAM) qui accordent à l'entité (par exemple, utilisateur, groupe ou rôle) l'autorisation d'effectuer une restauration. Sinon, il peut en résulter un comportement inattendu. Par exemple, supposons que vous ayez supprimé les autorisations d'écriture pour une table en cours de restauration. Dans ce cas, l'opération `RestoreTableToPointInTime` sous-jacente ne peut pas écrire les données restaurées dans la table. Les politiques IAM impliquant des restrictions d'adresses IP sources pour l'accès à la table de restauration cible peuvent également occasionner des problèmes. Vous pouvez modifier ou supprimer des autorisations uniquement lorsque l'opération de restauration est terminée.

Utilisation de AWS Backup avec DynamoDB

Amazon DynamoDB peut vous aider à répondre aux exigences de conformité réglementaire et de continuité d'activités grâce à des fonctions de sauvegarde améliorées dans AWS Backup. AWS Backup est un service de protection des données entièrement géré qui vous permet de centraliser et d'automatiser facilement les sauvegardes sur les services AWS, dans le cloud et sur site. À

l'aide de ce service, vous pouvez configurer des politiques de sauvegarde et surveiller l'activité de vos ressources AWS en un seul endroit. Pour utiliser AWS Backup, vous devez vous Inscrire positivement. Les choix d'inscription s'appliquent au compte et à la région AWS spécifiques. Il se peut donc que vous deviez vous inscrire à plusieurs régions en utilisant le même compte. Pour plus d'informations, consultez le Guide du développeur AWSBackup.

Amazon DynamoDB est intégré en natif à AWS Backup. Vous pouvez utiliser AWS Backup pour automatiquement planifier, copier et identifier vos sauvegardes à la demande DynamoDB, ainsi que pour établir leur cycle de vie. Vous pouvez continuer à afficher et à restaurer ces sauvegardes à partir de la console DynamoDB. Vous pouvez utiliser la console DynamoDB, l'API et l'interface de ligne de commande AWS (AWS CLI) pour activer les sauvegardes automatiques de vos tables DynamoDB.

> **Note**
>
> Toutes les sauvegardes effectuées via DynamoDB resteront inchangées. Vous serez toujours en mesure de créer des sauvegardes via le flux DynamoDB actuel.

Les fonctions de sauvegarde améliorées disponibles via AWS Backup incluent :

Sauvegardes planifiées - Vous pouvez configurer des sauvegardes régulièrement planifiées de vos tables DynamoDB à l'aide de plans de sauvegarde.

Copie entre comptes et entre régions - Vous pouvez copier automatiquement vos sauvegardes vers un autre coffre de sauvegarde dans une autre région ou un autre compte AWS, ce qui vous permet de prendre en charge vos exigences en matière de protection des données.

Hiérarchisation de stockage à froid - Vous pouvez configurer vos sauvegardes pour implémenter des règles de cycle de vie afin de supprimer ou de transférer des sauvegardes vers un stockage plus froid. Cela peut vous aider à optimiser vos coûts de sauvegarde.

Identifications - Vous pouvez identifier automatiquement vos sauvegardes à des fins de facturation et de répartition des coûts.

Chiffrement – Les sauvegardes à la demande DynamoDB gérées via AWS Backup sont maintenant stockées dans le coffre AWS Backup. Cela vous permet de chiffrer et de sécuriser vos sauvegardes à l'aide d'une clé AWS KMS key qui est indépendante de la clé de chiffrement de votre table DynamoDB.

Audit des sauvegardes – Vous pouvez utiliser AWS Backup Audit Manager pour vérifier la conformité de vos politiques AWS Backup et pour rechercher les activités de sauvegarde et les ressources qui ne sont pas encore conformes aux contrôles que vous avez définis. Vous pouvez également l'utiliser pour générer automatiquement une piste d'audit de rapports quotidiens et à la demande à des fins de gouvernance de sauvegarde.

Sauvegardes sécurisées à l'aide du modèle WORM – Vous pouvez utiliser le verrouillage de coffre AWS Backup pour activer un paramètre « Write Once Read Many » (WORM) pour vos sauvegardes. Avec le verrouillage de coffre d'AWS Backup, vous pouvez ajouter une couche de défense supplémentaire qui protège les sauvegardes contre les opérations de suppression involontaires ou malveillantes, les modifications apportées aux périodes de récupération des sauvegardes et les mises à jour des paramètres du cycle de vie. Pour en savoir plus, consultez le verrouillage de coffre AWS Backup.

Ces fonctions de sauvegarde améliorées sont disponibles dans toutes les régions AWS. Pour en savoir plus sur ces fonctions, veuillez consulter le AWS Backup Guide du développeur.

Rubriques

- Sauvegarde et restauration de tables DynamoDB AWS Backup avec : Comment ça marche
- Création de sauvegardes de tables DynamoDB avec AWS Backup
- Copie d'une sauvegarde d'une table DynamoDB avec AWS Backup
- Restauration d'une sauvegarde d'une table DynamoDB à partir d'AWS Backup
- Suppression d'une sauvegarde d'une table DynamoDB avec AWS Backup
- Différences d'utilisation entre les sauvegardes à la demande gérées par AWS Backup et DynamoDB

Sauvegarde et restauration de tables DynamoDB AWS Backup avec : Comment ça marche

Vous pouvez utiliser la fonction de sauvegarde à la demande pour créer des sauvegardes complètes de vos tables Amazon DynamoDB. Cette section présente le processus de sauvegarde et de restauration.

Sauvegardes

Lorsque vous créez une sauvegarde à la demande avec AWS Backup, un marqueur temporel de la demande est catalogué. La sauvegarde est créée de manière asynchrone par application de toutes les modifications jusqu'à l'heure de la demande sur l'instantané de la dernière table complète.

Chaque fois que vous créez une sauvegarde à la demande, toutes les données de la table sont sauvegardées. Il n'y a pas de limite au nombre de sauvegardes à la demande.

> **Note**
>
> Contrairement aux sauvegardes DynamoDB, les sauvegardes effectuées AWS Backup avec ne sont pas instantanées.

Lorsqu'une sauvegarde est en cours, vous ne pouvez pas effectuer les opérations suivantes :

- Suspendre ou annuler l'opération de sauvegarde.
- Supprimer la table source de la sauvegarde.
- Désactiver les sauvegardes d'une table si une sauvegarde de cette tables est en cours.

AWS Backup fournit des plannings de sauvegarde automatisés, une gestion de la rétention et une gestion du cycle de vie. Cela élimine le besoin de scripts personnalisés et de processus manuels. AWS Backup exécute les sauvegardes et les supprime lorsqu'elles expirent. Pour plus d'informations, consultez le Manuel du développeur AWS Backup.

Si vous utilisez la console, toutes les sauvegardes créées à l'aide de cette console AWS Backup sont répertoriées dans l'onglet Sauvegardes avec le type de sauvegarde défini sur`AWS_BACKUP`.

> **Note**
>
> Vous ne pouvez pas supprimer des sauvegardes marquées avec un Backup type (Type de sauvegarde) `AWS_BACKUP` à l'aide de la console DynamoDB. Pour gérer ces sauvegardes, utilisez la AWS Backup console.

Pour savoir comment effectuer une sauvegarde, consultez Sauvegarde d'une table DynamoDB.

Restaurations

Vous restaurez une table sans utiliser le débit provisionné pour cette table. Vous pouvez effectuer une restauration complète de la table à partir de votre sauvegarde DynamoDB ou configurer les paramètres de la table de destination. Lorsque vous effectuez une restauration, vous pouvez modifier les paramètres de table suivants :

- Paramètres de chiffrement

> **Important**
>
> Lorsque vous procédez à la restauration complète d'une table, la table de destination est définie avec les mêmes unités de capacité de lecture et d'écriture provisionnées que la table source avait au moment de la demande de sauvegarde. Le processus de restauration restaure également les index secondaires locaux et les index secondaires globaux.

Vous pouvez copier une sauvegarde des données de votre table DynamoDB dans une AWS autre région, puis la restaurer dans cette nouvelle région. Vous pouvez copier puis restaurer des sauvegardes entre les régions AWS commerciales, les régions de AWS Chine et les régions AWS GovCloud (États-Unis). Vous payez uniquement pour les données que vous copiez depuis la région source et pour les données que vous restaurez vers une nouvelle table dans la région de destination.

AWS Backup restaurera les tables avec tous les index d'origine.

Vous devez configurer manuellement les éléments suivants pour la table restaurée :

- Politiques de scalabilité automatique
- Gestion des identités et des accès AWS politiques (IAM)
- CloudWatch Mesures et alarmes Amazon
- Étiquettes
- Paramètres de flux
- Paramètres de time-to-live (TTL)
- Paramètres de protection contre la suppression
- Paramètres de récupération ponctuelle (PITR)

Vous pouvez restaurer uniquement les données de l'ensemble d'une table dans une nouvelle table à partir d'une sauvegarde. Vous ne pouvez écrire dans la table restaurée qu'une fois qu'elle est devenue active.

> **Note**
>
> AWS Backup les restaurations ne sont pas destructives. Vous ne pouvez pas remplacer une table existante au cours d'une opération de restauration.

Les métriques de service montrent que 95 % des restaurations de table des clients prennent moins d'une heure. Toutefois, les temps de restauration sont directement liés à la configuration de vos tables (taille et nombre de partitions sous-jacentes), ainsi qu'à d'autres variables connexes. Une bonne pratique de planification de la reprise après sinistre consiste à documenter régulièrement les temps moyens de restauration et à déterminer comment ces délais affectent votre objectif global de temps de récupération.

Pour savoir comment effectuer une restauration, consultez Restauration d'une table DynamoDB à partir d'une sauvegarde.

Vous pouvez utiliser des politiques IAM pour le contrôle d'accès. Pour de plus amples informations, veuillez consulter Utilisation d'IAM avec la sauvegarde et la restauration dans DynamoDB.

Toutes les actions de sauvegarde et de restauration de la console et de l'API sont capturées et enregistrées dans AWS CloudTrail pour la journalisation, la surveillance en continu et l'audit.

Création de sauvegardes de tables DynamoDB avec AWS Backup

Cette section décrit comment activer AWS Backup pour créer des sauvegardes à la demande et planifiées à partir de vos tables DynamoDB.

Rubriques

- Activation des fonctionnalités AWS Backup
- Sauvegardes à la demande
- Sauvegardes planifiées

Activation des fonctionnalités AWS Backup

Vous devez activer AWS Backup pour l'utiliser avec DynamoDB.

Pour activer AWS Backup, procédez comme suit :

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Dans la fenêtre Paramètres Backup, choisissez Activer.
4. Un écran de confirmation s'affiche. Choisissez Activer les fonctionnalités.

Les fonctions AWS Backup sont désormais disponibles pour vos tables DynamoDB.

Si vous choisissez de désactiver les fonctionnalités AWS Backup une fois qu'elles ont été activées, procédez comme suit :

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Dans la fenêtre Paramètres Backup, choisissez Désactiver.
4. Un écran de confirmation s'affiche. Choisissez Désactiver les fonctionnalités.

Si vous ne parvenez pas à activer ou à désactiver les fonctionnalités de AWS Backup, votre administrateur AWS peut avoir besoin d'effectuer ces actions.

Sauvegardes à la demande

Pour créer une sauvegarde à la demande d'une table DynamoDB, procédez comme suit :

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Choisissez Créer une sauvegarde.
4. Dans le menu déroulant qui s'affiche, choisissez Créer une sauvegarde à la demande.
5. Pour créer une sauvegarde gérée par AWS Backup avec un stockage à chaud et d'autres fonctions de base, choisissez Default Settings (Paramètres par défaut). Pour créer une sauvegarde pouvant être transférée vers un stockage à froid ou pour créer une sauvegarde avec des fonctions DynamoDB au lieu d'AWS Backup, choisissez Customize settings (Personnaliser les paramètres).

Si vous souhaitez plutôt créer cette sauvegarde avec des fonctions DynamoDB précédentes, choisissez Personnaliser les paramètres puis choisissez Sauvegarder avec DynamoDB.

6. Lorsque vous avez terminé le paramétrage, choisissez Créez une sauvegarde.

Sauvegardes planifiées

Pour planifier une sauvegarde, procédez comme suit.

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Dans le menu déroulant qui s'affiche, choisissez Planifier des sauvegardes avec AWS Backup.
4. Vous accéderez à AWS Backup pour créer un plan de sauvegarde.

Copie d'une sauvegarde d'une table DynamoDB avec AWS Backup

Vous pouvez effectuer une copie d'une sauvegarde en cours. Vous pouvez copier des sauvegardes vers plusieurs comptes AWS ou régions AWS à la demande ou automatiquement dans le cadre d'un plan de sauvegarde planifié. Vous pouvez également automatiser une séquence de copies entre comptes et entre régions pour Amazon DynamoDB Encryption Client.

La réplication entre régions est particulièrement utile en cas d'exigences de continuité d'activité ou de conformité pour stocker les sauvegardes à une distance minimale de vos données de production.

Les sauvegardes entre comptes sont utiles pour copier en toute sécurité vos sauvegardes vers un ou plusieurs comptes AWS de votre organisation pour des raisons opérationnelles ou de sécurité. Si votre sauvegarde d'origine est supprimée par inadvertance, vous pouvez copier la sauvegarde de son compte de destination vers son compte source, puis démarrer la restauration. Pour ce faire, vous devez disposer de deux comptes appartenant à la même organisation dans le service Organizations.

Les copies héritent de la configuration de la sauvegarde source, sauf indication contraire, à une exception près : si vous spécifiez que votre nouvelle copie n'expire « jamais ». Avec ce paramètre, la nouvelle copie hérite toujours de sa date d'expiration source. Si vous souhaitez que la nouvelle copie de votre sauvegarde soit permanente, définissez vos sauvegardes source pour qu'elles n'expirent jamais ou spécifiez que votre nouvelle copie expire 100 ans après sa création.

> **Note**
>
> Si vous effectuez une copie vers un autre compte, vous devez d'abord obtenir l'autorisation de ce compte.

Pour copier une sauvegarde, procédez comme suit :

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Cochez la case à côté de la sauvegarde que vous souhaitez copier.
   - Si la sauvegarde que vous souhaitez copier est grisée, vous devez activer Fonctionnalités avancées avec AWS Backup. Créez ensuite une nouvelle sauvegarde. Vous pouvez désormais copier cette nouvelle sauvegarde vers d'autres régions et comptes, et vous pourrez copier toutes les autres nouvelles sauvegardes à l'avenir.
4. Choisissez Copier.
5. Si vous souhaitez copier la sauvegarde vers un autre compte ou une autre région, cochez la case à côté de Copier le point de récupération vers une autre destination. Sélectionnez ensuite si vous souhaitez copier vers une autre région de votre compte ou vers un autre compte dans une autre région.

   > **Note**
   >
   > Pour restaurer une sauvegarde vers une autre région ou un autre compte, vous devez d'abord copier la sauvegarde vers cette région ou ce compte.

6. Sélectionnez le coffre souhaité dans lequel le fichier sera copié. Vous pouvez également créer un nouveau coffre de sauvegarde si vous le souhaitez.
7. Choisissez Copier la sauvegarde.

Restauration d'une sauvegarde d'une table DynamoDB à partir d'AWS Backup

Cette section décrit comment restaurer une sauvegarde d'une table DynamoDB à partir de AWS Backup.

Rubriques

- Restauration d'une table DynamoDB à partir d'AWS Backup
- Restauration d'une table DynamoDB vers une autre région ou un autre compte

Restauration d'une table DynamoDB à partir d'AWS Backup

Pour restaurer vos tables DynamoDB à partir de AWS Backup, procédez comme suit :

1. Connectez-vous à AWS Management Console (Console de gestion) et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez l'onglet Sauvegardes.
4. Cochez la case à côté de la sauvegarde précédente à partir de laquelle vous souhaitez effectuer la restauration.
5. Choisissez Restore (Restaurer). Vous accédez à l'écran Restaurer la table à partir de la sauvegarde.
6. Saisissez le nom de la table nouvellement restaurée, le chiffrement de cette nouvelle table, la clé avec laquelle vous souhaitez chiffrer la restauration et d'autres options.
7. Lorsque vous avez terminé, choisissez Restaurer.

Restauration d'une table DynamoDB vers une autre région ou un autre compte

Pour restaurer une table DynamoDB vers une autre région ou un autre compte, vous devez d'abord copier la sauvegarde vers cette nouvelle région ou ce nouveau compte. Pour pouvoir effectuer une copie vers un autre compte, ce compte doit d'abord vous accorder l'autorisation. Une fois que vous avez copié votre sauvegarde DynamoDB vers la nouvelle région ou le nouveau compte, elle peut être restaurée avec le processus de la section précédente.

Suppression d’une sauvegarde d’une table DynamoDB avec AWS Backup

Cette section décrit comment supprimer une sauvegarde d’une table DynamoDB avec AWS Backup.

Une sauvegarde DynamoDB créée via les fonctions AWS Backup est stockée dans un coffre AWS Backup.

Pour supprimer ce type de sauvegarde, procédez comme suit :

1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Sauvegardes.
3. Sur l'écran qui suit, choisissez Continuer vers AWS Backup.

   Vous accédez à Console AWS Backup. Pour en savoir plus sur la façon de supprimer des sauvegardes sur le Console AWS Backup, consultez Suppression de sauvegardes.

   Pour plus d'informations sur , consultez AWS BackupBackup and recovery using AWS Backup dans les AWSRecommandations.

Différences d'utilisation entre les sauvegardes à la demande gérées par AWS Backup et DynamoDB

Cette section décrit les différences techniques entre les sauvegardes à la demande gérées par AWS Backup et DynamoDB.

AWS Backup possède des flux et des comportements différents de ceux de DynamoDB. Il s'agit des licences suivantes :

Chiffrement - Les sauvegardes créées avec le plan AWS Backup sont stockées dans un coffre chiffré avec une clé gérée par le service AWS Backup. Le coffre dispose de politiques de contrôle d'accès pour plus de sécurité.

ARN de sauvegarde- Les fichiers de sauvegarde créés par AWS Backup disposeront maintenant d'un ARN AWS Backup, qui pourrait avoir un impact sur le modèle d'autorisation utilisateur. Les noms de ressources de sauvegarde (ARN) passeront de `arn:aws:dynamodb` à `arn:aws:backup`.

Suppression des sauvegardes - Les sauvegardes créées avecAWS Backup ne peuvent être supprimées qu'à partir du coffre AWS Backup. Vous ne serez pas en mesure de supprimer les fichiers AWS Backup de la console DynamoDB.

Processus de sauvegarde - Contrairement aux sauvegardes DynamoDB, les sauvegardes réalisées avec AWS Backup ne sont pas instantanées.

Facturation - Les sauvegardes de tables DynamoDB avec les fonctions AWS Backup sont facturées à partir de AWS Backup.

Rôles IAM - Si vous gérez l'accès via des rôles IAM, vous devez également configurer un nouveau rôle IAM avec ces nouvelles autorisations :

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` est nécessaire pour une sauvegarde réussie avec les fonctions AWS Backup, et `dynamodb:RestoreTableFromAwsBackup` est requis pour la restauration à partir d'une sauvegarde réalisée avec les fonctions AWS Backup.

Pour voir ces autorisations dans une politique IAM complète, consultez l'exemple 8 dans Utilisation d'IAM.

Exemples de code pour DynamoDB utilisant AWS SDKs

Les exemples de code suivants montrent comment utiliser DynamoDB avec AWS un kit de développement logiciel (SDK).

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

AWS les contributions communautaires sont des exemples qui ont été créés et sont maintenus par plusieurs équipes AWS. Pour faire un commentaire, utilisez le mécanisme fourni dans les référentiels liés.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Mise en route

Hello DynamoDB

Les exemples de code suivants montrent comment démarrer avec DynamoDB.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }
```

```
            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code pour le CMake fichier CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
```

```
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code pour le fichier source hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
```

```
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour C++ API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
```

```
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour plus de détails sur l'utilisation de DynamoDB AWS SDK pour JavaScript dans, consultez la section Programmation de DynamoDB avec. JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Pour plus de détails sur l'API, consultez ListTablesle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end
```

```
    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour Ruby API.

Exemples de code

- Exemples de base relatifs à l'utilisation de DynamoDB AWS SDKs
  - Hello DynamoDB
  - Découvrez les bases de DynamoDB avec un SDK AWS
  - Actions pour DynamoDB utilisant AWS SDKs
    - Utilisation BatchExecuteStatement avec un AWS SDK
    - Utilisation BatchGetItem avec un AWS SDK ou une CLI
    - Utilisation BatchWriteItem avec un AWS SDK ou une CLI
    - Utilisation CreateTable avec un AWS SDK ou une CLI
    - Utilisation DeleteItem avec un AWS SDK ou une CLI
    - Utilisation DeleteTable avec un AWS SDK ou une CLI
    - Utilisation DescribeTable avec un AWS SDK ou une CLI
    - Utilisation DescribeTimeToLive avec un AWS SDK ou une CLI
    - Utilisation ExecuteStatement avec un AWS SDK
    - Utilisation GetItem avec un AWS SDK ou une CLI
    - Utilisation ListTables avec un AWS SDK ou une CLI
    - Utilisation PutItem avec un AWS SDK ou une CLI

    - Utilisation Query avec un AWS SDK ou une CLI
    - Utilisation Scan avec un AWS SDK ou une CLI
    - Utilisation UpdateItem avec un AWS SDK ou une CLI
    - Utilisation UpdateTable avec un AWS SDK ou une CLI
    - Utilisation UpdateTimeToLive avec un AWS SDK ou une CLI
- Scénarios pour DynamoDB utilisant AWS SDKs
    - Accélérez les lectures DynamoDB avec DAX à l'aide d'un SDK AWS
    - Travaillez avec des scénarios d'index secondaire global DynamoDB avancés à l'aide de la version v2 AWS Command Line Interface
    - Créer une application pour soumettre des données à une table DynamoDB
    - Comparez plusieurs valeurs avec un seul attribut dans DynamoDB à l'aide d'un SDK AWS
    - Mettre à jour de manière conditionnelle un élément DynamoDB avec un TTL à l'aide d'un SDK AWS
    - Connectez-vous à une instance DynamoDB locale à l'aide d'un SDK AWS
    - Compter les opérateurs d'expression dans DynamoDB à l'aide d'un SDK AWS
    - Créer une API REST API Gateway pour suivre les données de la COVID-19
    - Créer une application de messagerie avec Step Functions
    - Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l’aide d’étiquettes
    - Création d'une table DynamoDB avec un index secondaire global à l'aide du SDK AWS
    - Création d'une table DynamoDB avec un paramètre de débit à chaud à l'aide d'un SDK AWS
    - Créer une application web pour suivre les données DynamoDB
    - Créer une application de chat Websocket avec API Gateway
    - Création d'un élément DynamoDB avec un TTL à l'aide d'un SDK AWS
    - Créez et gérez des tables globales DynamoDB avec une forte cohérence multirégionale à l'aide d'un SDK AWS
    - Création et gestion de tables globales DynamoDB illustrant MREC à l'aide d'un SDK AWS
    - Supprimer des données DynamoDB à l'aide des instructions PARTIQL DELETE avec un SDK AWS
    - Détectez le PPE dans les images avec Amazon Rekognition à l'aide d'un SDK AWS
    - Insérer des données DynamoDB à l'aide des instructions PartiQL INSERT avec un SDK AWS

- Invoquer une fonction Lambda à partir d'un navigateur
- Gestion des index secondaires globaux DynamoDB à l'aide de la version 2 AWS Command Line Interface
- Gérez les politiques basées sur les ressources DynamoDB à l'aide de la version 2 AWS Command Line Interface
- Surveillez les performances d'Amazon DynamoDB à l'aide d'un SDK AWS
- Exécuter des opérations de requête DynamoDB avancées à l'aide d'un SDK AWS
- Effectuez des opérations de liste dans DynamoDB à l'aide d'un SDK AWS
- Effectuez des opérations cartographiques dans DynamoDB à l'aide d'un SDK AWS
- Exécution d'opérations définies dans DynamoDB à l'aide d'un SDK AWS
- Interrogez une table DynamoDB à l'aide de lots d'instructions PartiQL et d'un SDK AWS
- Interroger une table DynamoDB à l'aide de PartiQL et d'un SDK AWS
- Interrogation d'une table DynamoDB à l'aide d'un index secondaire global avec un SDK AWS
- Interrogez une table DynamoDB à l'aide d'une condition begins_with avec un SDK AWS
- Interrogez une table DynamoDB en utilisant une plage de dates dans la clé de tri avec un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre dynamique à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre et limitez avec un SDK AWS
- Interrogez une table DynamoDB avec des attributs imbriqués à l'aide d'un SDK AWS
- Interroger une table DynamoDB avec pagination à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec des lectures très cohérentes à l'aide d'un SDK AWS
- Interrogez les données DynamoDB à l'aide des instructions PartiQL SELECT avec un SDK AWS
- Interrogez une table DynamoDB pour les éléments TTL à l'aide d'un SDK AWS
- Interrogez les tables DynamoDB à l'aide de modèles de date et d'heure avec un SDK AWS
- Enregistrez les informations EXIF et autres images à l'aide d'un SDK AWS
- Configuration du contrôle d'accès basé sur les attributs pour DynamoDB à l'aide de la version 2 AWS Command Line Interface
- Comprendre l'ordre des expressions de mise à jour dans DynamoDB à l'aide d'un SDK AWS
- Mettre à jour un paramètre de table DynamoDB avec un débit chaud à l'aide d'un SDK AWS
- Mettre à jour un élément DynamoDB avec un TTL à l'aide d'un SDK AWS

- Mettre à jour les données DynamoDB à l'aide des instructions PARTIQL UPDATE avec un SDK AWS
- Utiliser API Gateway pour invoquer une fonction Lambda
- Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda
- Utiliser un modèle de document pour DynamoDB à l'aide d'un SDK AWS
- Utiliser un modèle de persistance des objets de haut niveau pour DynamoDB à l'aide d'un SDK AWS
- Utiliser les opérations de compteur atomique dans DynamoDB avec un SDK AWS
- Utiliser des opérations conditionnelles dans DynamoDB avec un SDK AWS
- Utiliser les noms d'attributs d'expression dans DynamoDB avec un SDK AWS
- Utilisent des événements planifiés pour invoquer une fonction Lambda
- Utilisation des index secondaires locaux DynamoDB à l'aide de la version v2 AWS Command Line Interface
- Utilisation de DynamoDB Streams et utilisation de la version 2 Time-to-Live AWS Command Line Interface
- Utilisez les tables globales DynamoDB et la réplication multirégionale avec une cohérence éventuelle (MREC) à l'aide de la version v2 AWS Command Line Interface
- Utiliser le balisage des ressources DynamoDB à l'aide de la version v2 AWS Command Line Interface
- Utiliser le chiffrement des tables DynamoDB à l'aide de la version v2 AWS Command Line Interface

- Exemples sans serveur pour DynamoDB
  - Invocation d’une fonction Lambda à partir d’un déclencheur DynamoDB
  - Signalement des échecs d’articles par lots pour les fonctions Lambda à l’aide d’un déclencheur DynamoDB
- AWS contributions de la communauté pour DynamoDB
  - Création et test d’une application sans serveur

Exemples de base relatifs à l'utilisation de DynamoDB AWS SDKs

Les exemples de code suivants montrent comment utiliser les bases d'Amazon AWS SDKs DynamoDB avec.

Exemples

- Hello DynamoDB
- Découvrez les bases de DynamoDB avec un SDK AWS
- Actions pour DynamoDB utilisant AWS SDKs
  - Utilisation BatchExecuteStatement avec un AWS SDK
  - Utilisation BatchGetItem avec un AWS SDK ou une CLI
  - Utilisation BatchWriteItem avec un AWS SDK ou une CLI
  - Utilisation CreateTable avec un AWS SDK ou une CLI
  - Utilisation DeleteItem avec un AWS SDK ou une CLI
  - Utilisation DeleteTable avec un AWS SDK ou une CLI
  - Utilisation DescribeTable avec un AWS SDK ou une CLI
  - Utilisation DescribeTimeToLive avec un AWS SDK ou une CLI
  - Utilisation ExecuteStatement avec un AWS SDK
  - Utilisation GetItem avec un AWS SDK ou une CLI
  - Utilisation ListTables avec un AWS SDK ou une CLI
  - Utilisation PutItem avec un AWS SDK ou une CLI
  - Utilisation Query avec un AWS SDK ou une CLI
  - Utilisation Scan avec un AWS SDK ou une CLI
  - Utilisation UpdateItem avec un AWS SDK ou une CLI
  - Utilisation UpdateTable avec un AWS SDK ou une CLI
  - Utilisation UpdateTimeToLive avec un AWS SDK ou une CLI

Hello DynamoDB

Les exemples de code suivants montrent comment démarrer avec DynamoDB.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
```

```
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code pour le CMake fichier CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code pour le fichier source hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
```

```
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour C++ API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
```

```
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour plus de détails sur l'utilisation de DynamoDB AWS SDK pour JavaScript dans, consultez la section Programmation de DynamoDB avec. JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Pour plus de détails sur l'API, consultez ListTablesle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Découvrez les bases de DynamoDB avec un SDK AWS

Les exemples de code suivants montrent comment :

- Créez une table pouvant contenir des données vidéo.
- Insérer, récupérez et mettez à jour un seul film dans la table.
- Écrivez des données vidéo dans la table à partir d'un exemple de fichier JSON.
- Recherchez les films sortis au cours d'une année donnée.
- Recherchez les films sortis au cours d'une plage d'années spécifique.
- Supprimez un film de la table, puis supprimez la table.

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
```

```
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
 host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");
```

```
        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }
```

```
        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
```

```
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }
```

```
   /// <summary>
   /// Displays the description of the application on the console.
   /// </summary>
   private static void DisplayInstructions()
   {
       if (!IsInteractive)
       {
           return;
       }

       Console.Clear();
       Console.WriteLine();
       Console.Write(new string(' ', 28));
       Console.WriteLine("DynamoDB Basics Example");
       Console.WriteLine(SepBar);
       Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
       Console.WriteLine(SepBar);
       Console.WriteLine("The application does the following:");
       Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
       Console.WriteLine("\t2. Adds a single movie to the table.");
       Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
       Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
       Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
       Console.WriteLine("\t6. Deletes a movie.");
       Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
a given year.");
       Console.WriteLine("\t8. Uses ScanAsync to return all movies released
within a range of years.");
       Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
       WaitForEnter();
   }

   /// <summary>
   /// Simple method to wait for the Enter key to be pressed.
   /// </summary>
   private static void WaitForEnter()
   {
       if (IsInteractive)
       {
```

```
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Crée un tableau contenant des données vidéo.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

Ajoute un seul film à la table.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };
```

```
            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

Met à jour un seul élément d'une table.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
```

```
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
 {ex.Message}");
            throw;
        }
    }
```

Extrait un seul élément de la table des films.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
```

```
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Écrit un lot d'éléments dans la table des films.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
```

```
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
```

```
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Supprime un seul élément d'une table.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
```

```
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

Interroge la table des films sortis au cours d'une année donnée.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
```

```
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
```

```
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
 movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
 {ex.Message}");
            throw;
        }
    }
```

Recherche dans la table les films sortis au cours d'une plage d'années spécifique.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
```

```
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
 table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

Supprime la table des films.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
```

```
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Scénario de démarrage de DynamoDB.

```
###############################################################################
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {

  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
```

```
  get_input
  table_name=$get_input_result

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
\
    -k "$key_schema_json_file" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
  else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
  fi

  echo "Waiting for the table to become active...."

  if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
  else
    errecho "The table failed to become active. This demo will exit."
    cleanup "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo -n "Enter the title of a movie you want to add to the table: "
  get_input
  local added_title
  added_title=$get_input_result

  local added_year
  get_int_input "What year was it released? "
```

```
  added_year=$get_input_result

  local rating
  get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
  rating=$get_input_result

  local plot
  echo -n "Summarize the plot for me: "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"},
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
   }' >"$item_json_file"

  if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
    echo "The movie '$added_title' was successfully added to the table
'$table_name'."
  else
    errecho "Put item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "Let's update your movie '$added_title'."
  get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
  rating=$get_input_result

  echo -n "You summarized the plot as '$plot'."
  echo "What would you say now? "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"}
```

```
    }' >"$key_json_file"

  echo '{
    ":r": {"N" :"'"$rating"'"},
    ":p": {"S" : "'"$plot"'"}
   }' >"$item_json_file"

  local update_expression="SET info.rating = :r, info.plot = :p"

  if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
    echo "Updated '$added_title' with new attributes."
  else
    errecho "Update item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
```

```
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi

  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
      ask_for_year=false
    fi
```

```
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

  response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    errecho "Scan table failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo "Here is what I found:"
  echo "$response"

  echo
  echo_repeat "*" 88
  echo

  echo "Let's remove your movie '$added_title' from the table."

  if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
    echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"}
  }' >"$key_json_file"

    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
```

```
      errecho "Delete item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  fi

  if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
 then
    if ! clean_up "$table_name"; then
      return 1
    fi
  else
    if ! clean_up; then
      return 1
    fi
  fi

  return 0
}
```

Fonctions DynamoDB utilisées dans ce scénario.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.
```

```
 ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_create_table"
   echo "Creates an Amazon DynamoDB table with on-demand billing."
   echo " -n table_name  -- The name of the table to create."
   echo " -a attribute_definitions -- JSON file path of a list of attributes and
their types."
   echo " -k key_schema -- JSON file path of a list of attributes and their key
types."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:a:k:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
```

```
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
```

```
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}

###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }
```

```
  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
```

```
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }
```

```
while getopts "n:k:e:v:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    e) update_expression="${OPTARG}" ;;
    v) values="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
```

```
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

########################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
########################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
```

```
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$(
    aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text
  )
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports get-item operation failed.$response"
  return 1
fi

if [[ -n "$query" ]]; then
```

```
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
```

```
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
```

```
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
```

```
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
```

```
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
```

```
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
```

```
  echo ""
}
while getopts "n:k:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho ""

response=$(aws dynamodb delete-item \
  --table-name "$table_name" \
  --key file://"$keys")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans ce scénario.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table with partition: year (N) and sort: title (S).
(CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

            // 9. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
```

```
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie.
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
 it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::PutItemRequest putItemRequest;
        putItemRequest.SetTableName(MOVIE_TABLE_NAME);

        putItemRequest.AddItem(YEAR_KEY,

 Aws::DynamoDB::Model::AttributeValue().SetN(year));
        putItemRequest.AddItem(TITLE_KEY,

 Aws::DynamoDB::Model::AttributeValue().SetS(title));
```

```
        // Create attribute for the info map.
        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);

        putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

        Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
                putItemRequest);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add an item: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    // 3. Update the rating and plot of the movie by using an update expression.
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                           "'.\nWhat would you say now? ");

        Aws::DynamoDB::Model::UpdateItemRequest request;
        request.SetTableName(MOVIE_TABLE_NAME);
        request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
```

```
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        std::stringstream expressionStream;
        expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                         << INFO_KEY << "." << PLOT_KEY << " =:p";
        request.SetUpdateExpression(expressionStream.str());
        request.SetExpressionAttributeValues({
                                                 {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                         rating)},
                                                 {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                         plot)}
                                             });

        request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);

        const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
                request);
        if (!result.IsSuccess()) {
            std::cerr << "Error updating movie " + result.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    // 4. Put 250 movies in the table from moviedata.json.
    {
        std::cout << "Adding movies from a json file to the database." <<
std::endl;
        const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
        const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
        Aws::String jsonString = getMovieJSON();
        if (!jsonString.empty()) {
            Aws::Utils::Json::JsonValue json(jsonString);
            Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
            Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

            // To add movies with a cross-section of years, use an appropriate
increment
```

```
            // value for iterating through the database.
            size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
            for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
                writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                        movieJsons[i]);
                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(putItems);
                writeRequests.back().SetPutRequest(putRequest);
                if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                    Aws::DynamoDB::Model::BatchWriteItemRequest request;
                    request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                    const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                            request);
                    if (!outcome.IsSuccess()) {
                        std::cerr << "Unable to batch write movie data: "
                                  << outcome.GetError().GetMessage()
                                  << std::endl;
                        writeRequests.clear();
                        break;
                    }
                    else {
                        std::cout << "Added batch of " << writeRequests.size()
                                  << " movies to the database."
                                  << std::endl;
                    }
                    writeRequests.clear();
                }
            }
        }
    }

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    // 5. Get a movie by Key (partition + sort).
    {
        Aws::String titleToGet("King Kong");
        Aws::String answer = askQuestion(Aws::String(
                "Let's move on...Would you like to get info about '" + titleToGet
+
                "'? (y/n) "));
```

```
        if (answer == "y") {
            Aws::DynamoDB::Model::GetItemRequest request;
            request.SetTableName(MOVIE_TABLE_NAME);
            request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

            const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                    request);
            if (!result.IsSuccess()) {
                std::cerr << "Error " << result.GetError().GetMessage();
            }
            else {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
                if (!item.empty()) {
                    std::cout << "\nHere's what I found:" << std::endl;
                    printMovieInfo(item);
                }
                else {
                    std::cout << "\nThe movie was not found in the database."
                              << std::endl;
                }
            }
        }
    }

    // 6. Use Query with a key condition expression to return all movies
    //    released in a given year.
    Aws::String doAgain = "n";
    do {
        Aws::DynamoDB::Model::QueryRequest req;

        req.SetTableName(MOVIE_TABLE_NAME);

        // "year" is a DynamoDB reserved keyword and must be replaced with an
        // expression attribute name.
        req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
        req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

        int yearToMatch = askQuestionForIntRange(
```

```
                "\nLet's get a list of movies released in"
                " a given year. Enter a year between 1972 and 2018 ",
                1972, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
        attributeValues.emplace(":valueToMatch",
                                Aws::DynamoDB::Model::AttributeValue().SetN(
                                        yearToMatch));
        req.SetExpressionAttributeValues(attributeValues);

        const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
        if (result.IsSuccess()) {
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "\nThere were " << items.size()
                          << " movies in the database from "
                          << yearToMatch << "." << std::endl;
                for (const auto &item: items) {
                    printMovieInfo(item);
                }
                doAgain = "n";
            }
            else {
                std::cout << "\nNo movies from " << yearToMatch
                          << " were found in the database"
                          << std::endl;
                doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
            }
        }
        else {
            std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                      << std::endl;
        }

    } while (doAgain == "y");

    //  7. Use Scan to return movies released within a range of years.
    //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
    {
```

```
        int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                               "for movies in the database. Enter
a start year: ",
                                               1972, 2018);
        int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                             startYear, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
        do {
            Aws::DynamoDB::Model::ScanRequest scanRequest;
            scanRequest.SetTableName(MOVIE_TABLE_NAME);
            scanRequest.SetFilterExpression(
                    "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
            scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});

            Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
            attributeValues.emplace(":startYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            startYear));
            attributeValues.emplace(":endYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            endYear));
            scanRequest.SetExpressionAttributeValues(attributeValues);

            if (!exclusiveStartKey.empty()) {
                scanRequest.SetExclusiveStartKey(exclusiveStartKey);
            }

            const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                    scanRequest);
            if (result.IsSuccess()) {
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
                if (!items.empty()) {
                    std::stringstream stringStream;
                    stringStream << "\nFound " << items.size() << " movies in one
scan."
                                 << " How many would you like to see? ";
                    size_t count = askQuestionForInt(stringStream.str());
                    for (size_t i = 0; i < count && i < items.size(); ++i) {
```

```
                        printMovieInfo(items[i]);
                    }
                }
                else {
                    std::cout << "\nNo movies in the database between " <<
startYear <<
                              " and " << endYear << "." << std::endl;
                }

                exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
                if (!exclusiveStartKey.empty()) {
                    std::cout << "Not all movies were retrieved. Scanning for
more."
                              << std::endl;
                }
                else {
                    std::cout << "All movies were retrieved with this scan."
                              << std::endl;
                }
            }
            else {
                std::cerr << "Failed to Scan movies: "
                          << result.GetError().GetMessage() << std::endl;
            }
        } while (!exclusiveStartKey.empty());
    }

    // 8. Delete a movie. (DeleteItem)
    {
        std::stringstream stringStream;
        stringStream << "\nWould you like to delete the movie " << title
                     << " from the database? (y/n) ";
        Aws::String answer = askQuestion(stringStream.str());
        if (answer == "y") {
            Aws::DynamoDB::Model::DeleteItemRequest request;
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
            request.AddKey(TITLE_KEY,
                           Aws::DynamoDB::Model::AttributeValue().SetS(title));
            request.SetTableName(MOVIE_TABLE_NAME);

            const Aws::DynamoDB::Model::DeleteItemOutcome &result =
dynamoClient.DeleteItem(
                    request);
```

```
            if (result.IsSuccess()) {
                std::cout << "\nRemoved \"" << title << "\" from the database."
                          << std::endl;
            }
            else {
                std::cerr << "Failed to delete the movie: "
                          << result.GetError().GetMessage()
                          << std::endl;
            }
        }
    }

    return true;
}

//! Routine to convert a JsonView object to an attribute map.
/*!
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
```

```
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);
```

```
        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
```

```
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
```

```
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour C++.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation

- Analyser
- UpdateItem

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif pour créer la table et effectuer des actions dessus.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
```

```
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{TableName: tableName,
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
```

```
var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
```

```
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
```

```
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
 err = tableBasics.DeleteMovie(ctx, customMovie)
}
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
}

if questioner.AskBool("Delete the table, too? (y/n)", "y") {
 err = tableBasics.DeleteTable(ctx)
} else {
 log.Println("Don't forget to delete the table when you're done or you might " +
  "incur charges on your account.")
}
if err == nil {
 log.Printf("Deleted table %v.\n", tableBasics.TableName)
}
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
```

```
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Créez une structure et des méthodes qui appellent des actions DynamoDB.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
```

```
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
```

```
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
```

```
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
```

```
    }
   }
  }
  return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
```

```
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
```

```
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
```

```
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
```

```
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour Go.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une table DynamoDB.

```
// Create a table with a Sort key.
public static void createTable(DynamoDbClient ddb, String tableName) {
    DynamoDbWaiter dbWaiter = ddb.waiter();
    ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

    // Define attributes.
    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("year")
        .attributeType("N")
        .build());

    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("title")
        .attributeType("S")
        .build());

    ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
    KeySchemaElement key = KeySchemaElement.builder()
        .attributeName("year")
        .keyType(KeyType.HASH)
        .build();

    KeySchemaElement key2 = KeySchemaElement.builder()
        .attributeName("title")
        .keyType(KeyType.RANGE)
        .build();

    // Add KeySchemaElement objects to the list.
    tableKey.add(key);
    tableKey.add(key2);
```

```
        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Créez une fonction d'assistance pour télécharger et extraire l'exemple de fichier JSON.

```
    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
```

```
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }
```

Obtenez un élément d'une table.

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
```

```
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Exemple complet.

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
```

```
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        String tableName = "Movies";
        String fileName = "../../../resources/sample_files/movies.json";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
            "1. Creating an Amazon DynamoDB table named Movies with a key named
 year and a sort key named title.");
        createTable(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
 table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();
```

```
        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
```

```
            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            QueryConditional queryConditional = QueryConditional
                .keyEqualTo(Key.builder()
                    .partitionValue(2013)
                    .build());

            // Get items in the table and write out the ID value.
            Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
            String result = "";

            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The title of the movie is " +
rec.getTitle());
                System.out.println("The movie information  is " + rec.getInfo());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Scan the table.
    public static void scanMovies(DynamoDbClient ddb, String tableName) {
        System.out.println("******* Scanning all movies.\n");
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            Iterator<Movies> results = custTable.scan().items().iterator();
            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The movie title is " + rec.getTitle());
                System.out.println("The movie year is " + rec.getYear());
            }

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }

    // Update the record to include show only directors.
    public static void updateTableItem(DynamoDbClient ddb, String tableName) {
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
```

```
        itemKey.put("year", AttributeValue.builder().n("1933").build());
        itemKey.put("title", AttributeValue.builder().s("King Kong").build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("info", AttributeValueUpdate.builder()
            .value(AttributeValue.builder().s("{\"directors\":[\"Merian C. Cooper
\",\"Ernest B. Schoedsack\"]")
                .build())
            .action(AttributeAction.PUT)
            .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(itemKey)
            .attributeUpdates(updatedValues)
            .build();

        try {
            ddb.updateItem(request);
        } catch (ResourceNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        System.out.println("Item was updated!");
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
```

```
    }

    public static void putRecord(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

            // Populate the Table.
            Movies record = new Movies();
            record.setYear(2020);
            record.setTitle("My Movie2");
            record.setInfo("no info");
            table.putItem(record);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Added a new movie to the table.");
    }

    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
```

```
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";
```

```
const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });
```

```
  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
```

```
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
```

```
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
 * Query for movies by year.
 */

log("Querying for all movies from 1981.");
const paginatedQuery = paginateQuery(
  { client: docClient },
  {
    TableName: tableName,
    //For more information about query expressions, see
```

```
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
```

```
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une table DynamoDB.

```
suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
```

```
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

Créez une fonction d'assistance pour télécharger et extraire l'exemple de fichier JSON.

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}
```

```
suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Obtenez un élément d'une table.

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }
```

```
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

Exemple complet.

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
```

```
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}

// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
```

```
    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
```

```
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")
```

```
    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la AWS Référence de l'API de SDK pour Kotlin.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Interrogation
- Analyser
- UpdateItem

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Dans la mesure où cet exemple utilise des fichiers de support, veillez à lire les instructions contenues dans le fichier README.md contenant des exemples PHP.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();
```

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
$tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
|| $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
```

```
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
```

```
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
$rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
{$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
```

```
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour PHP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une classe qui encapsule une table DynamoDB.

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
```

```
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def exists(self, table_name):
    """
    Determines whether a table exists. As a side effect, stores the table in
    a member variable.

    :param table_name: The name of the table to check.
    :return: True when the table exists; otherwise, False.
    """
    try:
        table = self.dyn_resource.Table(table_name)
        table.load()
        exists = True
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            exists = False
        else:
            logger.error(
                "Couldn't check for existence of %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    else:
        self.table = table
    return exists
```

```
    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
```

```
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
```

```
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
```

```
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Créez une fonction d'assistance pour télécharger et extraire l'exemple de fichier JSON.

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.

    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
        raise
    else:
        # The sample file lists over 4000 movies, return only the first 250.
        return movie_data[:250]
```

Exécutez un scénario interactif pour créer la table et effectuer des actions dessus.

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
 table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
    print("-" * 88)

    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
 what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
```

```
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in
{release_year}:")
```

```
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
```

```
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

Ce scénario utilise la classe d'assistance suivante pour poser des questions à l'invite de commande.

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
```

```
        :param key: The key that is used for storing the answer in a dict, when
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
until
                           one fails or they all pass. Validators may also
convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators

    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
        When an answer fails validation, the complaint is printed and the
question
        is asked again.

        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
```

```
                answer, complaint = validator(answer)
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )
```

```
    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."

    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Interrogation
- Analyser
- UpdateItem

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une classe qui encapsule une table DynamoDB.

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
      { attribute_name: 'year', key_type: 'HASH' }, # Partition key
      { attribute_name: 'title', key_type: 'RANGE' } # Sort key
    ],
    attribute_definitions: [
      { attribute_name: 'year', attribute_type: 'N' },
      { attribute_name: 'title', attribute_type: 'S' }
    ],
    billing_mode: 'PAY_PER_REQUEST'
  )
  @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
  @table
rescue Aws::DynamoDB::Errors::ServiceError => e
  @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
```

```
    raise
  end
```

Créez une fonction d'assistance pour télécharger et extraire l'exemple de fichier JSON.

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        'https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip'
      )
      movie_json = ''
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

Exécutez un scénario interactif pour créer la table et effectuer des actions dessus.

```
  table_name = "doc-example-table-movies-#{rand(10**4)}"
  scaffold = Scaffold.new(table_name)
  dynamodb_wrapper = DynamoDBBasics.new(table_name)
```

```
 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Add a new record to the DynamoDB table.')
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask('Enter the title of a movie to add to
the table. E.g. The Matrix')
 my_movie[:year] = CLI::UI::Prompt.ask('What year was it released? E.g.
1989').to_i
 my_movie[:rating] = CLI::UI::Prompt.ask('On a scale of 1 - 10, how do you rate
it? E.g. 7').to_i
 my_movie[:plot] = CLI::UI::Prompt.ask('Enter a brief summary of the plot. E.g.
A man awakens to a new reality.')
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, 'Update a record in the DynamoDB table.')
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(4, 'Get a record from the DynamoDB table.')
 puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
 response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(5, 'Write a batch of items into the DynamoDB table.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
```

```
 print "Done!\n".green

 new_step(5, 'Query for a batch of items by key.')
 loop do
   release_year = CLI::UI::Prompt.ask('Enter a year between 1972 and 2018, e.g.
1999:').to_i
   results = dynamodb_wrapper.query_items(release_year)
   if results.any?
     puts("There were #{results.length} movies released in #{release_year}:")
     results.each do |movie|
       print "\t #{movie['title']}".green
     end
     break
   else
     continue = CLI::UI::Prompt.ask("Found no movies released in
#{release_year}! Try another year? (y/n)")
     break unless continue.eql?('y')
   end
 end
 print "\nDone!\n".green

 new_step(6, 'Scan for a batch of items using a filter expression.')
 years = {}
 years[:start] = CLI::UI::Prompt.ask('Enter a starting year between 1972 and
2018:')
 years[:end] = CLI::UI::Prompt.ask('Enter an ending year between 1972 and
2018:')
 releases = dynamodb_wrapper.scan_items(years)
 if !releases.empty?
   puts("Found #{releases.length} movies.")
   count = Question.ask(
     'How many do you want to see? ', method(:is_int), in_range(1,
releases.length)
   )
   puts("Here are your #{count} movies:")
   releases.take(count).each do |release|
     puts("\t#{release['title']}")
   end
 else
   puts("I don't know about any movies released between #{years[:start]} "\
        "and #{years[:end]}.")
 end
 print "\nDone!\n".green
```

```
  new_step(7, 'Delete an item from the DynamoDB table.')
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?('y')
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
    print "\nDone!\n".green
  end

  new_step(8, 'Delete the DynamoDB table.')
  answer = CLI::UI::Prompt.ask('Delete the table? (y/n)')
  if answer.eql?('y')
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts('Something went wrong with the demo.')
rescue Errno::ENOENT
  true
end
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour Ruby.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
```

```
        " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
```

```
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO /AWSEX/CL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
```

```
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour SAP ABAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Classe Swift qui gère les appels DynamoDB au SDK pour Swift.

```
import AWSDynamoDB
import Foundation

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
```

```
    case TableNotFound
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}

/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient?
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The optional Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
    /// `awaitTableActive()` to wait until the table's status is reported as
    /// ready to use by Amazon DynamoDB.
    ///
    init(region: String? = nil, tableName: String) async throws {
        do {
            let config = try await DynamoDBClient.DynamoDBClientConfiguration()
            if let region = region {
                config.region = region
            }

            self.ddbClient = DynamoDBClient(config: config)
            self.tableName = tableName

            try await self.createTable()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing Amazon
 DynamoDBClient client"))
```

```
            throw error
        }
    }

    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }


    /// Check to see if the table exists online yet.
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
```

```
    func tableExists() async throws -> Bool {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }

            return description.tableName == self.tableName
        } catch {
            print("ERROR: tableExists:", dump(error))
            throw error
        }
    }


    ///
    /// Waits for the table to exist and for its status to be active.
    ///
    func awaitTableActive() async throws {
        while try (await self.tableExists() == false) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }

        while try (await self.getTableStatus() != .active) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
```

```
        }
    }


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }


    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: self.tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }
            guard let status = description.tableStatus else {
                throw MoviesError.StatusUnknown
```

```
            }
            return status
        } catch {
            print("ERROR: getTableStatus:", dump(error))
            throw error
        }
    }


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }
```

```
            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
```

```
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws
    {
        do {
            let movie = Movie(title: title, year: year, rating: rating, plot:
plot)
            try await self.add(movie: movie)
        } catch {
            print("ERROR: add with fields:", dump(error))
            throw error
        }
    }


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
```

```
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }
```

```
            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
```

```
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
```

```
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
```

```
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }


    /// Delete a movie, given its title and release year.
    ///
```

```
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
}
```

Les structures utilisées par la MovieTable classe pour représenter les films.

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
```

```
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
```

```
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
{
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
```

```
                plot = plotVal
            }
        }

        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
}
```

Programme qui utilise la MovieTable classe pour accéder à une base de données DynamoDB.

```
import ArgumentParser
import ClientRuntime
import Foundation

import AWSDynamoDB


@testable import MovieList

extension String {
    // Get the directory if the string is a file path.
    func directory() -> String {
        guard let lastIndex = lastIndex(of: "/") else {
            print("Error: String directory separator not found.")
            return ""
        }
        return String(self[...lastIndex])
    }
}

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = #file.directory() + "../../../../../resources/
sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion: String?

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"
```

```
    /// Configuration details for the command.
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
DynamoDB!")

        //========================================================================
        // 1. Create the table. The Amazon DynamoDB table is represented by
        //    the `MovieTable` class.
        //========================================================================

        let tableName = "ddb-movies-sample-\(Int.random(in: 1 ... Int.max))"

        print("Creating table \"\(tableName)\"...")

        let movieDatabase = try await MovieTable(region: awsRegion,
                                                 tableName: tableName)

        print("\nWaiting for table to be ready to use...")
        try await movieDatabase.awaitTableActive()

        //========================================================================
        // 2. Add a movie to the table.
        //========================================================================

        print("\nAdding a movie...")
        try await movieDatabase.add(title: "Avatar: The Way of Water", year:
2022)
        try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

        //========================================================================
        // 3. Update the plot and rating of the movie using an update
        //    expression.
        //========================================================================
```

```
        print("\nAdding details to the added movie...")
        _ = try await movieDatabase.update(title: "Avatar: The Way of Water",
year: 2022,
                                           rating: 9.2, plot: "It's a sequel.")

        //===========================================================================
        // 4. Populate the table from the JSON file.
        //===========================================================================

        print("\nPopulating the movie database from JSON...")
        try await movieDatabase.populate(jsonPath: jsonPath)

        //===========================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //===========================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //===========================================================================
        // 6. Delete a movie.
        //===========================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
2022)

        //===========================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //===========================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
            print("    \(movie.title)")
        }
```

```
        //=============================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //=============================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
 lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }

        //=============================================================================
        // 9. Delete the table.
        //=============================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Swift.
  - BatchWriteItem
  - CreateTable
  - DeleteItem

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Interrogation
- Analyser
- UpdateItem

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Actions pour DynamoDB utilisant AWS SDKs

Les exemples de code suivants montrent comment effectuer des actions DynamoDB individuelles avec.AWS SDKs Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions pour configurer et exécuter le code.

Ces extraits appellent l'API DynamoDB et sont des extraits de code de programmes plus volumineux qui doivent être exécutés en contexte. Vous pouvez voir les actions dans leur contexte dans Scénarios pour DynamoDB utilisant AWS SDKs .

Les exemples suivants incluent uniquement les actions les plus couramment utilisées. Pour obtenir la liste complète, consultez la référence d'API d'Amazon DynamoDB.

Exemples

- Utilisation BatchExecuteStatement avec un AWS SDK
- Utilisation BatchGetItem avec un AWS SDK ou une CLI
- Utilisation BatchWriteItem avec un AWS SDK ou une CLI
- Utilisation CreateTable avec un AWS SDK ou une CLI
- Utilisation DeleteItem avec un AWS SDK ou une CLI
- Utilisation DeleteTable avec un AWS SDK ou une CLI
- Utilisation DescribeTable avec un AWS SDK ou une CLI
- Utilisation DescribeTimeToLive avec un AWS SDK ou une CLI
- Utilisation ExecuteStatement avec un AWS SDK

- Utilisation GetItem avec un AWS SDK ou une CLI
- Utilisation ListTables avec un AWS SDK ou une CLI
- Utilisation PutItem avec un AWS SDK ou une CLI
- Utilisation Query avec un AWS SDK ou une CLI
- Utilisation Scan avec un AWS SDK ou une CLI
- Utilisation UpdateItem avec un AWS SDK ou une CLI
- Utilisation UpdateTable avec un AWS SDK ou une CLI
- Utilisation UpdateTimeToLive avec un AWS SDK ou une CLI

Utilisation **`BatchExecuteStatement`** avec un AWS SDK

Les exemples de code suivants illustrent comment utiliser `BatchExecuteStatement`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Suppression de données à l'aide de PartiQL DELETE
- Insertion de données à l'aide de PartiQL INSERT
- Interrogation d'une table à l'aide de lots d'instructions PartiQL
- Interrogation des données à l'aide de PartiQL SELECT
- Mise à jour des données à l'aide de PartiQL UPDATE

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez des lots d'instructions INSERT pour ajouter des éléments.

```
        /// <summary>
```

```
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }
```

```
                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);
```

```
            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

Utilisez des lots d'instructions SELECT pour obtenir des éléments.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
```

```
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

Utilisez des lots d’instructions UPDATE pour mettre à jour des éléments.

```
        /// <summary>
```

```
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
```

```
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilisez des lots d'instructions DELETE pour supprimer des éléments.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
```

```
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez des lots d'instructions INSERT pour ajouter des éléments.

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
```

```
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
```

```
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

Utilisez des lots d'instructions SELECT pour obtenir des éléments.

```
    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();
```

```
            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Utilisez des lots d'instructions UPDATE pour mettre à jour des éléments.

```
    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";
```

```
        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Utilisez des lots d’instructions DELETE pour supprimer des éléments.

```
    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());
```

```
        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définissez une structure de réception de fonctions pour l'exemple.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Utilisez des lots d'instructions INSERT pour ajouter des éléments.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
```

```
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

Utilisez des lots d'instructions SELECT pour obtenir des éléments.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
```

```
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Utilisez des lots d'instructions UPDATE pour mettre à jour des éléments.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
```

```
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Utilisez des lots d'instructions DELETE pour supprimer des éléments.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour Go API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un lot d'éléments à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
```

```
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenez un lot d'éléments à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Mettez à jour un lot d'éléments à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Supprimez un lot d'éléments à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
```

```
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }
```

```
        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour PHP API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
```

```
                            each statement. This list must be in the same order as
the
                            statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Pour plus de détails sur l'API, consultez BatchExecuteStatementle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Lisez un lot d'éléments à l'aide de PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

Supprimez un lot d'éléments à l'aide de PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
```

```
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamodb = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamodb.table(table_name)
end

# Deletes a batch of items from a table using PartiQL
#
# @param batch_titles [Array] Collection of movie titles
# @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
def batch_execute_write(batch_titles)
  request_items = batch_titles.map do |title, year|
    {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
  end
  @dynamodb.client.batch_execute_statement({ statements: request_items })
end
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`BatchGetItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `BatchGetItem`.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void
 RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
```

```
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.
```

```
                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
```

```
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }

 Console.WriteLine("*********************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
```

```
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
```

```
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
#v");

    // Table1: Expression attribute names.
    Aws::Http::HeaderValueCollection headerValueCollection;
    headerValueCollection.emplace("#n", "Name");
    headerValueCollection.emplace("#v", "Views");
    table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

    // Table1: Set key name, type, and value to search.
    std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
    for (const Aws::String &name: nameValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetS(name);
        keys.emplace("Name", key);
        table1KeysAndAttributes.AddKeys(keys);
    }

    Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
    requestItems.emplace(table1Name, table1KeysAndAttributes);

    // Table2: ProductCatalog.
    Aws::String table2Name = "ProductCatalog";
    Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
    table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

    // Table2: Set key name, type, and value to search.
    std::vector<Aws::String> idValues = {"102", "103", "201"};
    for (const Aws::String &id: idValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetN(id);
        keys.emplace("Id", key);
        table2KeysAndAttributes.AddKeys(keys);
    }
```

```
   requestItems.emplace(table2Name, table2KeysAndAttributes);

   bool result = true;
   do {  // Use a do loop to handle pagination.
       request.SetRequestItems(requestItems);
       const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
               request);

       if (outcome.IsSuccess()) {
           for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
               Aws::String tableName = responsesMapEntry.first;
               const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
               std::cout << "Retrieved " << tableResults.size()
                         << " responses for table '" << tableName << "'.\n"
                         << std::endl;
               if (tableName == "Forum") {

                   std::cout << "Name | Category | Message | Views" <<
std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Name").GetS() << " | ";
                       std::cout << item.at("Category").GetS() << " | ";
                       std::cout << (item.count("Message") == 0 ? "" : item.at(
                               "Messages").GetN()) << " | ";
                       std::cout << (item.count("Views") == 0 ? "" : item.at(
                               "Views").GetN()) << std::endl;
                   }
               }
               else {
                   std::cout << "Title | Price | Color" << std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Title").GetS() << " | ";
                       std::cout << (item.count("Price") == 0 ? "" : item.at(
                               "Price").GetN());
                       if (item.count("Color")) {
                           std::cout << " | ";
                           for (const
std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
```

```
                                    "Color").GetL())
                                std::cout << listItem->GetS() << " ";
                        }
                        std::cout << std::endl;
                    }
                }
                std::cout << std::endl;
            }

            // If necessary, repeat request for remaining items.
            requestItems = outcome.GetResult().GetUnprocessedKeys();
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour extraire plusieurs éléments d'une table

L'exemple `batch-get-items` suivant lit plusieurs éléments de la table `MusicCollection` à l'aide d'un lot de trois demandes `GetItem`, et demande le nombre d'unités de capacité de lecture consommées par l'opération. La commande renvoie uniquement l'attribut `AlbumTitle`.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Contenu de `request-items.json` :

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

Sortie :

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
```

```
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Pour plus d'informations, consultez Opérations par lots dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Montre comment obtenir des éléments par lots à l'aide du client de service.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();
```

```
        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
 dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
 batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
 responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
        }
    }
}
```

Montre comment obtenir des éléments par lots à l'aide du client de service et d'un programme de pagination.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>
```

```
                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir BatchGet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
```

```
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};
```

```
ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : extrait l'élément SongTitle « Somewhere Down The Road » des tables DynamoDB « Music » et « Songs ».

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
```

```
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : extrait l'élément SongTitle « Somewhere Down The Road » des tables DynamoDB « Music » et « Songs ».

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
```

```
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import decimal
import json
import logging
import os
import pprint
```

```
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.


def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.

    :param batch_keys: The set of keys to retrieve. A batch can contain at most
 100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
```

```
                "%s unprocessed keys returned. Sleep, then retry.",
 unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break

    return retrieved
```

- Pour plus de détails sur l'API, consultez BatchGetItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Gets an array of `Movie` objects describing all the movies in the
    /// specified list. Any movies that aren't found in the list have no
    /// corresponding entry in the resulting array.
    ///
    /// - Parameters
    ///     - keys: An array of tuples, each of which specifies the title and
    ///       release year of a movie to fetch from the table.
    ///
    /// - Returns:
```

```
    ///     - An array of `Movie` objects describing each match found in the
    ///     table.
    ///
    /// - Throws:
    ///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
    ///     been initialized.
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MovieError.ClientUninitialized
            }

            var movieList: [Movie] = []
            var keyItems: [[Swift.String: DynamoDBClientTypes.AttributeValue]] =
[]

            // Convert the list of keys into the form used by DynamoDB.

            for key in keys {
                let item: [Swift.String: DynamoDBClientTypes.AttributeValue] = [
                    "title": .s(key.title),
                    "year": .n(String(key.year))
                ]
                keyItems.append(item)
            }

            // Create the input record for `batchGetItem()`. The list of
requested
            // items is in the `requestItems` property. This array contains one
            // entry for each table from which items are to be fetched. In this
            // example, there's only one table containing the movie data.
            //
            // If we wanted this program to also support searching for matches
            // in a table of book data, we could add a second `requestItem`
            // mapping the name of the book table to the list of items we want to
            // find in it.
            let input = BatchGetItemInput(
                requestItems: [
                    self.tableName: .init(
                        consistentRead: true,
                        keys: keyItems
                    )
                ]
```

```
            )

            // Fetch the matching movies from the table.

            let output = try await client.batchGetItem(input: input)

            // Get the set of responses. If there aren't any, return the empty
            // movie list.

            guard let responses = output.responses else {
                return movieList
            }

            // Get the list of matching items for the table with the name
            // `tableName`.

            guard let responseList = responses[self.tableName] else {
                return movieList
            }

            // Create `Movie` items for each of the matching movies in the table
            // and add them to the `MovieList` array.

            for response in responseList {
                try movieList.append(Movie(withItem: response))
            }

            return movieList
        } catch {
            print("ERROR: batchGet", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`BatchWriteItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `BatchWriteItem`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Principes de base

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Écrit un lot d'éléments dans la table des films.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
```

```
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
```

```
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
##############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);
```

```
   if (!fileStream) {
       std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                 << std::endl;
   }

   std::stringstream stringStream;
   stringStream << fileStream.rdbuf();
   Aws::Utils::Json::JsonValue jsonValue(stringStream);

   Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
   Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
jsonValue.View().GetAllObjects();
   for (const auto &level1Entry: level1Map) {
       const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
       const Aws::String &tableName = level1Entry.first;
       // The JSON entries at this level are as follows:
       //  key - table name
       //  value - list of request objects
       if (!entriesView.IsListType()) {
           std::cerr << "Error: JSON file entry '"
                     << tableName << "' is not a list." << std::endl;
           continue;
       }

       Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
entriesView.AsArray();

       Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
       if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                              writeRequests)) {
           batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
       }
   }

   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
dynamoClient.BatchWriteItem(
           batchWriteItemRequest);

   if (outcome.IsSuccess()) {
       std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
   }
```

```
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return outcome.IsSuccess();
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
```

```
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
```

```
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour ajouter plusieurs éléments à une table

L'exemple `batch-write-item` suivant ajoute trois nouveaux éléments à la table `MusicCollection` à l'aide d'un lot de trois demandes `PutItem`. Il demande également des informations sur le nombre d'unités de capacité d'écriture consommées par l'opération et sur les collections d'éléments modifiées par l'opération.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Contenu de `request-items.json` :

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
```

```
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

Sortie :

```
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
```

```
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

Pour plus d'informations, consultez Opérations par lots dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
```

```
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Insère de nombreux éléments dans une table à l'aide du client de service.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """
```

```
                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
song!").build());
        item1Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle1").build());

writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
melody.").build());
        item2Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle2").build());
```

```
 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri

        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Insère de nombreux éléments dans une table à l'aide du client amélioré.

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import java.time.ZoneOffset;

/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
        }

        public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
 {
                try {
                        DynamoDbTable<Customer> customerMappedTable =
 enhancedClient.table("Customer",
                                        TableSchema.fromBean(Customer.class));
                        DynamoDbTable<Music> musicMappedTable =
 enhancedClient.table("Music",
                                        TableSchema.fromBean(Music.class));
                        LocalDate localDate = LocalDate.parse("2020-04-07");
                        LocalDateTime localDateTime = localDate.atStartOfDay();
```

```
                        Instant instant =
localDateTime.toInstant(ZoneOffset.UTC);

                        Customer record2 = new Customer();
                        record2.setCustName("Fred Pink");
                        record2.setId("id110");
                        record2.setEmail("fredp@noserver.com");
                        record2.setRegistrationDate(instant);

                        Customer record3 = new Customer();
                        record3.setCustName("Susan Pink");
                        record3.setId("id120");
                        record3.setEmail("spink@noserver.com");
                        record3.setRegistrationDate(instant);

                        Customer record4 = new Customer();
                        record4.setCustName("Jerry orange");
                        record4.setId("id101");
                        record4.setEmail("jorange@noserver.com");
                        record4.setRegistrationDate(instant);

                        BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                        .builder()
                                        .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)

.addPutItem(builder -> builder.item(record2))

.addPutItem(builder -> builder.item(record3))

.addPutItem(builder -> builder.item(record4))
                                                                        .build(),

WriteBatch.builder(Music.class) // delete an item from the Music

      // table

.mappedTableResource(musicMappedTable)
```

```
 .addDeleteItem(builder -> builder.key(

       Key.builder().partitionValue(

                        "Famous Band")

                        .build()))
                                                                    .build())
                                      .build();

                        // Add three items to the Customer table and delete one
 item from the Music
                        // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                        System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
```

```
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour JavaScript API.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
        $total = 0;
```

```
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
 ['Item' => $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : crée un nouvel élément ou remplace un élément existant par un nouveau dans les tables DynamoDB Music et Songs.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem
```

```
$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Sortie :

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : crée un nouvel élément ou remplace un élément existant par un nouveau dans les tables DynamoDB Music et Songs.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Sortie :

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour plus de détails sur l'API, consultez BatchWriteItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
  #                            the keys required by the schema that was specified
 when the
  #                            table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({ put_request: { item: movie } })
      end
      @dynamo_resource.client.batch_write_item({ request_items: { @table.name =>
 movie_items } })
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:"
```

```
    )
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour Ruby API.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
 jsonData)
```

```
            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
```

```
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`CreateTable`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `CreateTable`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Scénarios d'index secondaire global avancés
- Création d'une table avec un index secondaire global
- Création d'une table avec le débit à chaud activé
- Création et gestion de tables globales démontrant MREC
- Création et gestion des tables globales MRSC
- Gestion des index secondaires globaux
- Gestion des politiques basées sur les ressources
- Configuration d'un contrôle d'accès par attributs
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)
- Utilisation d'index secondaires locaux
- Utilisation du balisage des ressources
- Travaillez avec Streams et Time-to-Live
- Utilisation du chiffrement des tables

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour créer une table avec des balises

L'exemple `create-table` suivant utilise les attributs et le schéma de clés spécifiés pour créer une table nommée `MusicCollection`. Cette table utilise le débit provisionné et est chiffrée au repos à l'aide de la clé CMK AWS détenue par défaut. La commande applique également une balise à la table, avec une clé `Owner` et une valeur `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
```

```
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour créer une table en mode à la demande

L'exemple suivant crée une table appelée `MusicCollection` en mode à la demande, plutôt qu'en mode débit provisionné. Cela est utile pour les tables dont les charges de travail sont imprévisibles.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```


Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour créer une table et la chiffrer à l'aide d'une clé CMK gérée par le client

L'exemple suivant crée une table nommée `MusicCollection` et la chiffre à l'aide d'une clé CMK gérée par le client.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
```

```
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour créer une table avec un index secondaire local

L'exemple suivant utilise les attributs et le schéma de clés spécifiés pour créer une table nommée `MusicCollection` avec un index local secondaire nommé `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
```

```
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 5 : pour créer une table avec un index secondaire global

L'exemple suivant crée une table nommée `GameScores` avec un index secondaire global nommé `GameTitleIndex`. La table de base a une clé de partition `UserId` et une clé de tri `GameTitle`, vous permettant de trouver efficacement le meilleur score d'un utilisateur pour un jeu spécifique, tandis que l'index secondaire global (GSI) a une clé de partition `GameTitle` et

une clé de tri `TopScore`, vous permettant de trouver rapidement le score le plus élevé pour un jeu particulier.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
```

```
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
```

```
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 6 : pour créer une table avec plusieurs index secondaires globaux à la fois

L'exemple suivant crée une table nommée `GameScores` avec deux index secondaires globaux. Les schémas GSI sont transmis via un fichier plutôt que sur la ligne de commande.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Contenu de `gsi.json` :

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
```

```
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
```

```
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
```

```
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 7 : pour créer une table avec Streams activé

L'exemple suivant crée une table appelée `GameScores` avec DynamoDB Streams activé. Les nouvelles et les anciennes images de chaque élément seront écrites dans le flux.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
```

```
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Pour plus d'informations, consultez Opérations de base pour les tables dans le Guide du développeur Amazon DynamoDB.

Exemple 8 : pour créer une table avec Keys-Only Stream activé

L'exemple suivant crée une table appelée `GameScores` avec DynamoDB Streams activé. Seuls les attributs clés des éléments modifiés sont écrits dans le flux.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
```

```
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Sortie :


```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
```

```
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Pour plus d'informations, consultez Modifier la récupération de données pour DynamoDB Streams dans le Guide du développeur Amazon DynamoDB.

Exemple 9 : pour créer une table à l'aide de la classe Standard Infrequent Access

L'exemple suivant crée une table appelée `GameScores` et attribue la classe de table Standard-Infrequent Access (DynamoDB Standard-IA). Cette classe de table est optimisée pour le stockage, qui constitue le principal coût.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
```

```
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Pour plus d'informations, consultez Classes de tables dans le Guide du développeur Amazon DynamoDB.

Exemple 10 : pour créer une table avec la protection contre la suppression activée

L'exemple suivant crée une table appelée `GameScores` et active la protection contre la suppression.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
```

```
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Pour plus d'informations, consultez Utilisation de la protection contre la suppression dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
```

```
   public static void main(String[] args) {
       final String usage = """

           Usage:
               <tableName> <key>

           Where:
               tableName - The Amazon DynamoDB table to create (for example,
Music3).
               key - The key for the Amazon DynamoDB table (for example,
Artist).
           """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
           .region(region)
           .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
           .attributeDefinitions(AttributeDefinition.builder()
               .attributeName(key)
               .attributeType(ScalarAttributeType.S)
               .build())
           .keySchema(KeySchemaElement.builder()
               .attributeName(key)
               .keyType(KeyType.HASH)
               .build())
```

```
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
```

```
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
```

```
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Pour plus de détails sur l'API, consultez CreateTablela section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créer une table.

```
        $tableName = "ddb_demo_table_$uuid";
```

```
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Le

schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemple 2 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Un index secondaire local est également défini. La clé de l'index secondaire local sera définie automatiquement à partir de la clé de hachage principale de la table (ForumName). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
```

```
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemple 3 : Cet exemple montre comment utiliser un pipeline unique pour créer une table nommée Thread qui possède une clé primaire composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés) et un index secondaire local. Les options Add- DDBKey Schema et Add- DDBIndex Schema créent un nouvel TableSchema objet pour vous si aucun objet n'est fourni par le pipeline ou le paramètre -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l'aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemple 2 : Cet exemple crée une table nommée Thread dont la clé primaire est composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés). Un index secondaire local est également défini. La clé de l'index secondaire local sera définie automatiquement à partir de la clé de hachage principale de la table (ForumName). Le schéma utilisé pour construire la table peut être redirigé vers chaque applet de commande comme indiqué ou spécifié à l’aide du paramètre -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemple 3 : Cet exemple montre comment utiliser un pipeline unique pour créer une table nommée Thread qui possède une clé primaire composée de « ForumName » (hachage de type de clé) et de « Subject » (plage de types de clés) et un index secondaire local. Les options Add- DDBKey Schema et Add- DDBIndex Schema créent un nouvel TableSchema objet pour vous si aucun objet n'est fourni par le pipeline ou le paramètre -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Sortie :

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une table pour stocker des données vidéo.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
```

```
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Pour plus de détails sur l'API, consultez CreateTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
```

```
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Pour plus de détails sur l'API, voir CreateTablela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
```

```
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez CreateTablela section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
```

```
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`DeleteItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DeleteItem`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Utilisation d'opérations conditionnelles

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };
```

```
            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
##############################################################################
```

```
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1
```

```
  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
```

```
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#########################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#########################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#########################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#########################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
```

```
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;
```

```
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
```

```
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour supprimer un élément

L'exemple `delete-item` suivant supprime un élément de la table `MusicCollection` et demande des informations sur l'élément supprimé et sur la capacité utilisée par la demande.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Sortie :

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour supprimer un élément sous certaines conditions

L'exemple suivant supprime un élément de la table `ProductCatalog` uniquement si `ProductCategory` est `Sporting Goods` ou `Gardening Supplies` et si son prix est compris entre 500 et 600. Elle renvoie des informations sur l'élément qui a été supprimé.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenu de `names.json` :

```
{
    "#P": "Price"
}
```

Contenu de `values.json` :

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Sortie :

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
```

```
    }
}
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}
```

```
// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }
```

```
    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir DeleteCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez un élément d'une table.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
```

```
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Supprimez un élément d'une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun deleteDynamoDBItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        DeleteItemRequest {
            tableName = tableNameVal
            key = keyToGet
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Pour plus de détails sur l'API, consultez DeleteItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : supprime l'élément DynamoDB correspondant à la clé fournie.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : supprime l'élément DynamoDB correspondant à la clé fournie.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.
```

```
    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
```

Vous pouvez spécifier une condition pour qu'un élément soit supprimé uniquement lorsqu'il répond à certains critères.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- Pour plus de détails sur l'API, consultez DeleteItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- Pour plus de détails sur l'API, voir DeleteItemla section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez DeleteItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB
```

```
    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`DeleteTable`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DeleteTable`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Création et gestion des tables globales MRSC
- Gestion des index secondaires globaux
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
```

```
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour supprimer une table

L'exemple `delete-table` suivant supprime la table `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Sortie :

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Pour plus d'informations, consultez Suppression d'une table dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
```

```
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
```

```
        println("$tableNameVal was deleted")
    }
}
```

- Pour plus de détails sur l'API, consultez DeleteTablela section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : supprime la table spécifiée. Vous êtes invité à confirmer avant que l'opération ne se poursuive.

```
Remove-DDBTable -TableName "myTable"
```

Exemple 2 : supprime la table spécifiée. Aucune confirmation ne vous est demandée avant le début de l'opération.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : supprime la table spécifiée. Vous êtes invité à confirmer avant que l'opération ne se poursuive.

```
Remove-DDBTable -TableName "myTable"
```

Exemple 2 : supprime la table spécifiée. Aucune confirmation ne vous est demandée avant le début de l'opération.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
```

```
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour plus de détails sur l'API, consultez DeleteTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Pour plus de détails sur l'API, voir DeleteTablela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
```

```
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez DeleteTablela section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
```

```
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`DescribeTable`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DescribeTable`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Création et gestion de tables globales démontrant MREC
- Création et gestion des tables globales MRSC
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)
- Travaillez avec Streams et Time-to-Live
- Utilisation du chiffrement des tables

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    private static async Task GetTableInformation()
    {
        Console.WriteLine("\n*** Retrieving table information ***");

        var response = await Client.DescribeTableAsync(new DescribeTableRequest
        {
            TableName = ExampleTableName
        });

        var table = response.Table;
        Console.WriteLine($"Name: {table.TableName}");
        Console.WriteLine($"# of items: {table.ItemCount}");

    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
```

```
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
```

```
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::DescribeTableRequest request;
   request.SetTableName(tableName);

   const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
 dynamoClient.DescribeTable(
           request);

   if (outcome.IsSuccess()) {
       const Aws::DynamoDB::Model::TableDescription &td =
 outcome.GetResult().GetTable();
       std::cout << "Table name  : " << td.GetTableName() << std::endl;
       std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
       std::cout << "Status      : "
                 <<
 Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                         td.GetTableStatus()) << std::endl;
       std::cout << "Item count  : " << td.GetItemCount() << std::endl;
       std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;

       const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
 td.GetProvisionedThroughput();
       std::cout << "Throughput" << std::endl;
       std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
 std::endl;
       std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
 std::endl;

       const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
 td.GetAttributeDefinitions();
       std::cout << "Attributes" << std::endl;
       for (const auto &a: ad)
           std::cout << "  " << a.GetAttributeName() << " (" <<

 Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                           a.GetAttributeType()) <<
                   ")" << std::endl;
   }
```

```
    else {
        std::cerr << "Failed to describe table: " <<
 outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour décrire un tableau

L'exemple `describe-table` suivant décrit la table `MusicCollection`.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

Sortie :

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

Pour plus d'informations, consultez Description d'une table dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information
 about (for example, Music3).
                """;

        if (args.length != 1) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Getting description for %s\n\n", tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        describeDymamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void describeDymamoDBTable(DynamoDbClient ddb, String
tableName) {
        DescribeTableRequest request = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            TableDescription tableInfo = ddb.describeTable(request).table();
            if (tableInfo != null) {
                System.out.format("Table name  : %s\n", tableInfo.tableName());
                System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
                System.out.format("Status      : %s\n", tableInfo.tableStatus());
                System.out.format("Item count  : %d\n", tableInfo.itemCount());
                System.out.format("Size (bytes): %d\n",
tableInfo.tableSizeBytes());

                ProvisionedThroughputDescription throughputInfo =
tableInfo.provisionedThroughput();
                System.out.println("Throughput");
                System.out.format("  Read Capacity : %d\n",
throughputInfo.readCapacityUnits());
                System.out.format("  Write Capacity: %d\n",
throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
```

```
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : renvoie les détails de la table spécifiée.

```
Get-DDBTable -TableName "myTable"
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : renvoie les détails de la table spécifiée.

```
Get-DDBTable -TableName "myTable"
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
```

```
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
                raise
        else:
            self.table = table
        return exists
```

- Pour plus de détails sur l'API, consultez DescribeTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
```

```
rescue Aws::DynamoDB::Errors::ResourceNotFoundException
  @logger.debug("Table #{table_name} doesn't exist")
  false
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't check for existence of #{table_name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
ENDTRY.
```

- Pour plus de détails sur l'API, consultez DescribeTablela section de référence du AWS SDK pour l'API SAP ABAP.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`DescribeTimeToLive`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DescribeTimeToLive`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Travaillez avec Streams et Time-to-Live

CLI

AWS CLI

Pour afficher les paramètres Durée de vie (TTL) d'une table

L'exemple `describe-time-to-live` suivant affiche les paramètres de durée de vie de la table `MusicCollection`.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Sortie :

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Pour plus d'informations, consultez Time to Live dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous DescribeTimeToLiveà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

Décrire la configuration Durée de vie (TTL) sur une table DynamoDB existante à l'aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.logging.Level;
import java.util.logging.Logger;

    public DescribeTimeToLiveResponse describeTTL(final String tableName, final
 Region region) {
        final DescribeTimeToLiveRequest request =
            DescribeTimeToLiveRequest.builder().tableName(tableName).build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.describeTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeTimeToLiveà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Décrivez la configuration Durée de vie (TTL) sur une table DynamoDB existante à l'aide du kit AWS SDK pour JavaScript.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const describeTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED')
 {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Pour plus de détails sur l'API, reportez-vous DescribeTimeToLiveà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Décrivez la configuration Durée de vie (TTL) sur une table DynamoDB existante à l'aide du kit AWS SDK pour Python (Boto3).

```
import boto3


def describe_ttl(table_name, region):
    """
    Describes TTL on an existing table, as well as a region.

    :param table_name: String representing the name of the table
    :param region: AWS Region of the table - example `us-east-1`
    :return: Time to live description.
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        ttl_description = dynamodb.describe_time_to_live(TableName=table_name)
        print(
            f"TimeToLive for table {table_name} is status
 {ttl_description['TimeToLiveDescription']['TimeToLiveStatus']}"
        )

        return ttl_description
    except Exception as e:
        print(f"Error describing table: {e}")
        raise


# Enter your own table name and AWS region
describe_ttl("your-table-name", "us-east-1")
```

- Pour plus de détails sur l'API, consultez DescribeTimeToLivele AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`ExecuteStatement`** avec un AWS SDK

Les exemples de code suivants illustrent comment utiliser `ExecuteStatement`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Suppression de données à l'aide de PartiQL DELETE
- Insertion de données à l'aide de PartiQL INSERT
- Interrogation d'une table à l'aide de PartiQL
- Interrogation des données à l'aide de PartiQL SELECT
- Mise à jour des données à l'aide de PartiQL UPDATE

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez une instruction INSERT pour ajouter un élément.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilisez une instruction SELECT pour obtenir un élément.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });
```

```
            return response.Items;
        }
```

Utilisez une instruction SELECT pour obtenir une liste d'éléments.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Utilisez une instruction UPDATE pour mettre à jour un élément.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
```

```
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilisez une instruction DELETE pour supprimer un seul film.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
```

```
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez une instruction INSERT pour ajouter un élément.

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
```

```
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
```

```
                    << std::endl;
            return false;
        }
    }
```

Utilisez une instruction SELECT pour obtenir un élément.

```
   //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           // Print the retrieved movie information.
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
```

```
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }
```

Utilisez une instruction UPDATE pour mettre à jour un élément.

```
    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
```

```
        }
    }
```

Utilisez une instruction DELETE pour supprimer un élément.

```
    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définissez une structure de réception de fonctions pour l'exemple.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Utilisez une instruction INSERT pour ajouter un élément.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
```

```
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Utilisez une instruction SELECT pour obtenir un élément.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
```

```
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Utilisez une instruction SELECT pour obtenir une liste d'éléments et projeter les résultats.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
```

```
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

Utilisez une instruction UPDATE pour mettre à jour un élément.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Utilisez une instruction DELETE pour supprimer un élément.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
```

```
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
```

```
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour Go API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

Créez un élément à l’aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenez un élément à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
```

```
  return response;
};
```

Mettez à jour d'un élément à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Supprimez un élément à l'aide de PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
```

```
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }
```

```
    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour PHP API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
   def run_partiql(self, statement, params):
       """
       Runs a PartiQL statement. A Boto3 resource is used even though
       `execute_statement` is called on the underlying `client` object because
the
       resource transforms input and output from plain old Python objects
(POPOs) to
       the DynamoDB format. If you create the client directly, you must do these
       transforms yourself.

       :param statement: The PartiQL statement.
       :param params: The list of PartiQL parameters. These are applied to the
                      statement in the order they are listed.
       :return: The items returned from the statement, if any.
       """
       try:
           output = self.dyn_resource.meta.client.execute_statement(
               Statement=statement, Parameters=params
           )
       except ClientError as err:
           if err.response["Error"]["Code"] == "ResourceNotFoundException":
               logger.error(
                   "Couldn't execute PartiQL '%s' because the table does not
exist.",
                   statement,
               )
           else:
               logger.error(
                   "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                   statement,
                   err.response["Error"]["Code"],
                   err.response["Error"]["Message"],
               )
           raise
       else:
           return output
```

- Pour plus de détails sur l'API, consultez ExecuteStatementle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Sélectionnez un seul élément à l'aide de PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Mettez à jour un seul élément à l'aide de PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
```

```
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Ajoutez un seul élément à l'aide de PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
```

```
      parameters: [title, year, { 'plot': plot, 'rating': rating }]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Supprimez un seul élément à l'aide de PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def delete_item_by_title(title, year)
    request = {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`GetItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `GetItem`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Création et gestion des tables globales MRSC
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
```

```
            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$(
    aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text
  )
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports get-item operation failed.$response"
  return 1
fi
```

```
  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
```

```
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
```

```
                                const Aws::String &partitionKey,
                                const Aws::String &partitionValue,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour lire un élément dans une table

L'exemple `get-item` suivant récupère un élément de la table `MusicCollection`. La table possède une clé hash-and-range primaire (`ArtistetSongTitle`), vous devez donc spécifier ces deux attributs. La commande demande également des informations sur la capacité de lecture consommée par l'opération.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Sortie :

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
```

```
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour lire un élément en utilisant une lecture cohérente

L'exemple suivant récupère un élément à partir de la table `MusicCollection` à l'aide de la lecture fortement cohérente.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Sortie :

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour extraire des attributs spécifiques d'un élément

L'exemple suivant utilise une expression de projection pour extraire uniquement trois attributs de l'élément souhaité.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Contenu de `names.json` :

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Sortie :

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Pour plus d'informations, consultez Lecture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
```

```
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
```

```
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Récupère un élément d'une table à l'aide du DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
```

```
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir un élément d'une table

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenez un élément d'une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
```

```
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Pour plus de détails sur l'API, consultez GetItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : renvoie l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : renvoie l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem
```

```
Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
```

```
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Pour plus de détails sur l'API, consultez GetItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        oo_item = lo_dyn->getitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key ).
        DATA(lt_attr) = oo_item->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez GetItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB
```

```
    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`ListTables`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `ListTables`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Configuration d'un contrôle d'accès par attributs

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    private static async Task ListMyTables()
    {
        Console.WriteLine("\n*** Listing tables ***");

        string lastTableNameEvaluated = null;
        do
        {
            var response = await Client.ListTablesAsync(new ListTablesRequest
            {
                Limit = 2,
                ExclusiveStartTableName = lastTableNameEvaluated
            });

            foreach (var name in response.TableNames)
            {
                Console.WriteLine(name);
            }
```

```
            lastTableNameEvaluated = response.LastEvaluatedTableName;
        } while (lastTableNameEvaluated != null);
    }
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi
```

```
  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
```

```
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
```

```
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }

        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
                outcome.GetResult().GetLastEvaluatedTableName());

    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour répertorier les tables

L'`list-tables`exemple suivant répertorie toutes les tables associées au AWS compte courant et à la région.

```
aws dynamodb list-tables
```

Sortie :

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
```

```
}
```

Pour plus d'informations, consultez Liste des noms de tables dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour limiter la taille de page

L'exemple suivant renvoie une liste de toutes les tables existantes, mais extrait un seul élément par appel, en effectuant plusieurs appels si nécessaire pour obtenir la liste complète. La limitation de la taille de page est utile lorsque vous exécutez des commandes de liste sur un grand nombre de ressources, ce qui peut entraîner une erreur « expiré » lors de l'utilisation du format de page par défaut de 1 000.

```
aws dynamodb list-tables \
    --page-size 1
```

Sortie :

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Pour plus d'informations, consultez Liste des noms de tables dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour limiter le nombre d'éléments retournés

L'exemple suivant limite le nombre d'éléments renvoyés à 2. La réponse inclut une valeur `NextToken` permettant d'extraire la page suivante des résultats.

```
aws dynamodb list-tables \
    --max-items 2
```

Sortie :

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Pour plus d'informations, consultez Liste des noms de tables dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour récupérer la page de résultats suivante

La commande suivante utilise la valeur `NextToken` d'un appel précédent à la commande `list-tables` pour récupérer une autre page de résultats. Dans ce cas, comme la réponse n'inclut aucune valeur `NextToken`, nous savons que nous avons atteint la fin des résultats.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Sortie :

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Pour plus d'informations, consultez Liste des noms de tables dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
```

```
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
```

```
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d’informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun listAllTables() {
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Pour plus de détails sur l'API, consultez ListTablesla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : renvoi des détails de toutes les tables, en itérant automatiquement jusqu'à ce que le service indique qu'il n'existe plus de tables.

```
Get-DDBTableList
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : renvoi des détails de toutes les tables, en itérant automatiquement jusqu'à ce que le service indique qu'il n'existe plus de tables.

```
Get-DDBTableList
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.
```

```
    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- Pour plus de détails sur l'API, consultez ListTablesle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Déterminez si une table existe.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
```

```
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

Déterminez si une table existe.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

- Pour plus de détails sur l'API, voir ListTablesla section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez ListTableslа section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Get a list of the DynamoDB tables available in the specified Region.
    ///
    /// - Returns: An array of strings listing all of the tables available
    ///   in the Region specified when the session was created.
    public func getTableList() async throws -> [String] {
        let input = ListTablesInput(
        )
        return try await session.listTables(input: input)
    }
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`PutItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `PutItem`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Scénarios d'index secondaire global avancés
- Création d'un élément avec un TTL
- Création et gestion de tables globales démontrant MREC
- Création et gestion des tables globales MRSC
- Utilisation d'opérations conditionnelles
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)
- Travaillez avec Streams et Time-to-Live

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
```

```
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
```

```
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$item" ]]; then
  errecho "ERROR: You must provide an item with the -i parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    item:   $item"
iecho ""
iecho ""

response=$(aws dynamodb put-item \
  --table-name "$table_name" \
  --item file://"$item")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports put-item operation failed.$response"
  return 1
fi

return 0
```

```
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
```

```
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
```

```
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
```

```
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
```

```
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour ajouter un élément à une table

L'`put-item`exemple suivant ajoute un nouvel élément au MusicCollectiontableau.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenu de `item.json` :

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Sortie :

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
```

```
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour remplacer un élément d'une table sous certaines conditions

L'exemple `put-item` suivant remplace un élément existant dans la table `MusicCollection` uniquement si celui-ci possède un attribut `AlbumTitle` dont la valeur est `Greatest Hits`. La commande renvoie la valeur précédente de l'élément.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenu de `item.json` :

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Contenu de `names.json` :

```
{
    "#A": "AlbumTitle"
}
```

Contenu de `values.json` :

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Sortie :

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

Si la clé existe déjà, vous devriez voir la sortie suivante :

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
```

```
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
```

```
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Place un élément dans un tableau en utilisant DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
 Happy Day).
                **Warning** This program will  place an item that you specify
 into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
```

```
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Insérez un élément dans la table.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
```

```
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Insérez un élément dans une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
```

```
    }
}
```

- Pour plus de détails sur l'API, consultez PutItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
```

```
        $this->dynamoDbClient->putItem($array);
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : création d'un nouvel élément ou remplacement d'un élément existant par un nouvel élément.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : création d'un nouvel élément ou remplacement d'un élément existant par un nouvel élément.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour plus de détails sur l'API, consultez PutItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
```

```
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Pour plus de détails sur l'API, voir PutItemla section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Pour plus de détails sur l'API, consultez PutItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
```

```
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`Query`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `Query`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Scénarios d'index secondaire global avancés
- Comparaison de plusieurs valeurs avec un seul attribut
- Gestion des index secondaires globaux
- Exécution d'opérations de requête avancées
- Interrogation d'une table à l'aide d'une condition begins_with
- Interrogation d'une table à l'aide d'une plage de dates
- Interrogation d'une table à l'aide d'un index secondaire global
- Interrogation d'une table avec une expression de filtre complexe
- Interrogation d'une table avec une expression de filtre dynamique
- Interrogation d'une table avec une expression de filtre et des limites
- Interrogation d'une table avec des attributs imbriqués
- Interrogation d'une table avec pagination
- Interrogation d'une table avec des lectures fortement cohérentes
- Interrogation des éléments TTL
- Interrogation de tables à l'aide de modèles de date et d'heure
- Utilisation des noms d'attributs d'expression
- Utilisation d'index secondaires locaux

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
```

```
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour .NET.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
```

```
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi
```

```
  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Pour plus d'informations sur l'API, consultez Query dans la Référence des commandes AWS CLI.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
```

```
                                    const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::QueryRequest request;

   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
```

```
                              <<
"*************************************************"
                              << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour C++.

CLI

AWS CLI

Exemple 1 : pour interroger une table

L'exemple `query` suivant interroge des éléments dans la table `MusicCollection`. La table possède une clé hash-and-range primaire (`ArtistetSongTitle`), mais cette requête indique uniquement la valeur de la clé de hachage. Elle renvoie les titres des chansons de l'artiste « No One You Know ».

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Pour plus d'informations, consultez Utilisation de requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour interroger une table à l'aide de lectures fortement cohérentes et parcourir l'index par ordre décroissant

L'exemple suivant exécute la même requête que le premier exemple, mais renvoie les résultats dans l'ordre inverse et utilise des lectures fortement cohérentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour filtrer des résultats spécifiques

L'exemple suivant interroge `MusicCollection`, mais exclut les résultats contenant des valeurs spécifiques dans l'attribut `AlbumTitle`. Notez que cela n'affecte pas `ScannedCount` ni `ConsumedCapacity`, car le filtre est appliqué après la lecture des éléments.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Contenu de `values.json` :

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenu de `names.json` :

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Sortie :

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
```

```
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour extraire uniquement le nombre d'éléments

L'exemple suivant extrait le nombre d'éléments correspondant à la requête, mais n'extrait aucun des éléments eux-mêmes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

Exemple 5 : pour interroger un index

L'exemple suivant interroge l'index secondaire global `AlbumTitleIndex`. La requête renvoie tous les attributs de la table de base qui ont été projetés dans l'index secondaire local. Notez que lorsque vous interrogez un index secondaire local ou un index secondaire global, vous devez également fournir le nom de la table de base à l'aide du paramètre `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Contenu de `expression-attributes.json` :

```
{
    ":v1": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
```

```
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Pour plus d'informations, consultez Utilisation des requêtes dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

- Pour plus d'informations sur l'API, consultez Query dans la Référence des commandes AWS CLI.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
```

```
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
```

```
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour Go.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Interroge une table à l'aide de DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
```

```
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
```

```
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Interroge une table à l'aide de `DynamoDbClient` et d'un index secondaire.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
```

```
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK pour Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
```

```
        return $this->dynamoDbClient->query($query);
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour PHP.

PowerShell

Outils pour PowerShell V4

Exemple 1 : invoque une requête qui renvoie des éléments DynamoDB avec les valeurs spécifiées et Artist. SongTitle

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des applets de commande pour les Outils AWS pour PowerShell (V4).

Outils pour PowerShell V5

Exemple 1 : invoque une requête qui renvoie des éléments DynamoDB avec les valeurs spécifiées et Artist. SongTitle

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des applets de commande pour les Outils AWS pour PowerShell (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Interrogez des éléments à l'aide d'une expression de condition clé.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return response["Items"]
```

Interrogez des éléments et projetez-les pour renvoyer un sous-ensemble de données.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
```

```
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour Ruby.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Trouvez les films réalisés au cours de l'année spécifiée.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;
```

```
    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Pour plus de détails sur l'API, consultez Query dans la Référence des API du kit AWS SDK pour Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
```

```
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Pour plus d'informations sur l'API, consultez Query dans le guide de référence d'API du kit SDK AWS pour SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
```

```
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence de l'API du kit SDK AWS for Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`Scan`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `Scan`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Accélérer les lectures avec DAX
- Comparaison de plusieurs valeurs avec un seul attribut

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
```

```
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour .NET.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
```

```
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1
```

```
 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$filter_expression" ]]; then
   errecho "ERROR: You must provide a filter expression with the -f parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_names" ]]; then
   errecho "ERROR: You must provide expression attribute names with the -a
parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_values" ]]; then
   errecho "ERROR: You must provide expression attribute values with the -v
parameter."
   usage
   return 1
 fi

 if [[ -z "$projection_expression" ]]; then
   response=$(aws dynamodb scan \
     --table-name "$table_name" \
     --filter-expression "$filter_expression" \
     --expression-attribute-names file://"$expression_attribute_names" \
     --expression-attribute-values file://"$expression_attribute_values")
 else
   response=$(aws dynamodb scan \
     --table-name "$table_name" \
     --filter-expression "$filter_expression" \
     --expression-attribute-names file://"$expression_attribute_names" \
     --expression-attribute-values file://"$expression_attribute_values" \
     --projection-expression "$projection_expression")
 fi

 local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
```

```
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus d'informations sur l'API, consultez Scan dans la Référence des commandes AWS CLI.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::ScanRequest request;
   request.SetTableName(tableName);

   if (!projectionExpression.empty())
       request.SetProjectionExpression(projectionExpression);

   Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
 all_items;
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 last_evaluated_key; // Used for pagination;
   do {
       if (!last_evaluated_key.empty()) {
           request.SetExclusiveStartKey(last_evaluated_key);
       }
       const Aws::DynamoDB::Model::ScanOutcome &outcome =
 dynamoClient.Scan(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           all_items.insert(all_items.end(), items.begin(), items.end());

           last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
       }
       else {
           std::cerr << "Failed to Scan items: " <<
 outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }

   } while (!last_evaluated_key.empty());

   if (!all_items.empty()) {
       std::cout << "Number of items retrieved from scan: " << all_items.size()
                 << std::endl;
       // Iterate each item and print.
       for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
```

```
            std::cout << "*****************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour C++.

CLI

AWS CLI

Pour analyser une table

L'exemple `scan` suivant analyse l'intégralité de la table `MusicCollection`, puis limite les résultats aux chansons de l'artiste « No One You Know ». Pour chaque élément, seuls le titre de l'album et le titre de la chanson sont renvoyés.

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

Contenu de `expression-attribute-names.json` :

```
{
```

```
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

Contenu de `expression-attribute-values.json` :

```
{
    ":a": {"S": "No One You Know"}
}
```

Sortie :

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

Pour plus d'informations, consultez Utilisation des analyses dans DynamoDB dans le Guide du développeur Amazon DynamoDB.

- Pour plus d'informations sur l'API, consultez Scan dans la Référence des commandes AWS CLI.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
```

```
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour de plus amples informations sur l'API, consultez Analyser dans la référence d'API AWS SDK pour Go.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Analyse une table Amazon DynamoDB à l'aide de. DynamoDbClient

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
```

```
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
```

```
                    System.out.println("The value is " + item.get(key).s());
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir ScanCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });
```

```
  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour JavaScript.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Analyser dans la référence d'API AWS SDK pour JavaScript.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
```

```
            }
        }
    }
}
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
```

```
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour PHP.

PowerShell

Outils pour PowerShell V4

Exemple 1 : renvoi de tous les éléments de la table Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Exemple 2 : renvoie les éléments du tableau Musique dont le score est CriticRating supérieur ou égal à neuf.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des applets de commande pour les Outils AWS pour PowerShell (V4).

Outils pour PowerShell V5

Exemple 1 : renvoi de tous les éléments de la table Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
```

```
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Exemple 2 : renvoie les éléments du tableau Musique dont le score est CriticRating supérieur ou égal à neuf.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des cmdlet Outils AWS pour PowerShell(V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class Movies:
```

```
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
```

```
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- Pour obtenir plus de détails sur l'API, consultez Scan dans la référence des API du kit AWS SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
def initialize(table_name)
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Scans for movies that were released in a range of years.
# Uses a projection expression to return a subset of data for each movie.
#
# @param year_range [Hash] The range of years to retrieve.
# @return [Array] The list of movies released in the specified years.
def scan_items(year_range)
  movies = []
  scan_hash = {
    filter_expression: '#yr between :start_yr and :end_yr',
    projection_expression: '#yr, title, info.rating',
    expression_attribute_names: { '#yr' => 'year' },
    expression_attribute_values: {
      ':start_yr' => year_range[:start], ':end_yr' => year_range[:end]
    }
  }
  done = false
  start_key = nil
  until done
    scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
    response = @table.scan(scan_hash)
    movies.concat(response.items) unless response.items.empty?
    start_key = response.last_evaluated_key
    done = start_key.nil?
  end
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't scan for movies. Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
else
  movies
end
```

- Pour de plus amples informations sur l'API, consultez Analyser dans la référence d'API AWS SDK pour Ruby.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }

    Ok(())
}
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Pour plus d'informations sur l'API, consultez Scan dans le guide de référence d'API du kit SDK AWS pour SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
 = nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
```

```
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }
```

- Pour plus d'informations sur l'API, consultez Scan dans la référence de l'API du kit SDK AWS pour Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`UpdateItem`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `UpdateItem`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Principes de base
- Mise à jour sous certaines conditions du TTL d'un élément
- Comptage des opérateurs d'expression
- Création et gestion des tables globales MRSC
- Exécution d'opérations de liste
- Exécution d'opérations de mappage
- Exécution d'opérations d'ensemble
- Compréhension de l'ordre des expressions de mise à jour
- Mise à jour du TTL d'un élément
- Utilisation d'opérations de compteurs atomiques
- Utilisation d'opérations conditionnelles
- Utilisation des noms d'attributs d'expression

.NET

SDK pour .NET(v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };
```

```
            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour .NET API.

Bash

AWS CLI avec le script Bash

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
###############################################################################
# function dynamodb_update_item
```

```
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho "    update_expression:   $update_expression"
iecho "    values:   $values"

response=$(aws dynamodb update-item \
  --table-name "$table_name" \
  --key file://"$keys" \
  --update-expression "$update_expression" \
  --expression-attribute-values file://"$values")
```

```
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Fonctions utilitaires utilisées dans cet exemple.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS CLI commandes.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);
```

```
    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
```

```
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour mettre à jour un élément dans une table

L'exemple `update-item` suivant met à jour un élément dans la table `MusicCollection`. Il ajoute un nouvel attribut (`Year`) et modifie l'attribut `AlbumTitle`. Tous les attributs de l'élément, tels qu'ils apparaissent après la mise à jour, sont renvoyés dans la réponse.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenu de `expression-attribute-names.json` :

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contenu de `expression-attribute-values.json` :

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Sortie :

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```


Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour mettre à jour un élément sous certaines conditions

L'exemple suivant met à jour un élément de la table `MusicCollection`, mais uniquement si l'élément existant ne possède pas encore d'attribut `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Contenu de `key.json` :

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenu de `expression-attribute-names.json` :

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenu de `expression-attribute-values.json` :

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Si l'élément possède déjà un attribut `Year`, DynamoDB renvoie le résultat suivant.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Pour plus d'informations, consultez Écriture d'un élément dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS CLI commandes.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
```

```
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Met à jour un élément d'un tableau à l'aide de DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
```

```
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
example, Awards).
                    updateVal - The value used to update an item (for example,
14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
```

```
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
```

```
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Pour plus de détails sur l'API, consultez UpdateItemla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
```

```
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour PHP API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : définit l'attribut genre sur « Rap » sur l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
```

```
$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Sortie :

```
Name                           Value
----                           -----
Genre                          Rap
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : définit l'attribut genre sur « Rap » sur l'élément DynamoDB avec la clé de partition et la SongTitle clé de tri Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Sortie :

```
Name                           Value
```

```
----                           -----
Genre                          Rap
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Mettez à jour un élément à l'aide d'une expression de mise à jour.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
```

```
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Mettez à jour un élément à l'aide d'une expression de mise à jour qui inclut une opération arithmétique.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Mettre à jour un élément uniquement lorsqu'il remplit certaines conditions.

```
class UpdateQueryWrapper:
```

```
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
greater
        than a specified threshold. If the movie does not list more than the
threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Attributes"]
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
 #{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
```

```
end
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       oo_output = lo_dyn->updateitem(
         iv_tablename        = iv_table_name
         it_key              = it_item_key
         it_attributeupdates = it_attribute_updates ).
       MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
     CATCH /aws1/cx_dyncondalcheckfaile00.
       MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
     CATCH /aws1/cx_dynresourcenotfoundex.
       MESSAGE 'The table or index does not exist' TYPE 'E'.
     CATCH /aws1/cx_dyntransactconflictex.
       MESSAGE 'Another transaction is using the item' TYPE 'E'.
   ENDTRY.
```

- Pour plus de détails sur l'API, consultez UpdateItemla section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
 nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
 [:]

            if rating != nil {
```

```
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`UpdateTable`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `UpdateTable`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action dans son contexte dans les exemples de code suivants :

- Création et gestion de tables globales démontrant MREC
- Création et gestion des tables globales MRSC
- Gestion des index secondaires globaux
- Mise à jour du paramètre de débit chaud d'une table
- Utilisation des tables globales et la cohérence à terme de la réplication multirégionale (MREC)
- Utilisation du chiffrement des tables

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Update a DynamoDB table.
/*!
  \sa updateTable()
  \param tableName: Name for the DynamoDB table.
  \param readCapacity: Provisioned read capacity.
  \param writeCapacity: Provisioned write capacity.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::updateTable(const Aws::String &tableName,
```

```
                                    long long readCapacity, long long
writeCapacity,
                                    const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   std::cout << "Updating " << tableName << " with new provisioned throughput
values"
             << std::endl;
   std::cout << "Read capacity : " << readCapacity << std::endl;
   std::cout << "Write capacity: " << writeCapacity << std::endl;

   Aws::DynamoDB::Model::UpdateTableRequest request;
   Aws::DynamoDB::Model::ProvisionedThroughput provisionedThroughput;

provisionedThroughput.WithReadCapacityUnits(readCapacity).WithWriteCapacityUnits(
           writeCapacity);

request.WithProvisionedThroughput(provisionedThroughput).WithTableName(tableName);

   const Aws::DynamoDB::Model::UpdateTableOutcome &outcome =
dynamoClient.UpdateTable(
           request);
   if (outcome.IsSuccess()) {
       std::cout << "Successfully updated the table." << std::endl;
   } else {
       const Aws::DynamoDB::DynamoDBError &error = outcome.GetError();
       if (error.GetErrorType() == Aws::DynamoDB::DynamoDBErrors::VALIDATION &&
           error.GetMessage().find("The provisioned throughput for the table
will not change") != std::string::npos) {
           std::cout << "The provisioned throughput for the table will not
change." << std::endl;
       } else {
           std::cerr << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
   }

   return waitTableActive(tableName, dynamoClient);
}
```

Code qui attend que la table soit active.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Exemple 1 : pour modifier le mode de facturation d'une table

L'exemple `update-table` suivant augmente la capacité de lecture et d'écriture provisionnée sur la table `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
```

```
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
    }
}
```

Pour plus d'informations, consultez Mise à jour d'une table dans le Guide du développeur Amazon DynamoDB.

Exemple 2 : pour créer un index secondaire global

L'exemple suivant ajoute un index secondaire global à la table `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Contenu de `gsi-updates.json` :

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
```

```
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
```

```
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
```

```
            }
        ]
    }
}
```

Pour plus d'informations, consultez Mise à jour d'une table dans le Guide du développeur Amazon DynamoDB.

Exemple 3 : pour activer DynamoDB Streams sur une table

La commande suivante active DynamoDB Streams sur la table `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
```

```
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
}
```

Pour plus d'informations, consultez Mise à jour d'une table dans le Guide du développeur Amazon DynamoDB.

Exemple 4 : pour activer le chiffrement côté serveur

L'exemple suivant active le chiffrement côté serveur sur la table `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Sortie :

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
```

```
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
```

```
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
            "Status": "UPDATING"
        }
    }
}
```

Pour plus d'informations, consultez Mise à jour d'une table dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : mise à jour du débit provisionné pour la table donnée.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : mise à jour du débit provisionné pour la table donnée.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation **`UpdateTimeToLive`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `UpdateTimeToLive`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Travaillez avec Streams et Time-to-Live

CLI

AWS CLI

Pour mettre à jour les paramètres Durée de vie d'une table

L'exemple `update-time-to-live` suivant active la durée de vie au niveau de la table spécifiée.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Sortie :

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

Pour plus d'informations, consultez Time to Live dans le Guide du développeur Amazon DynamoDB.

- Pour plus de détails sur l'API, reportez-vous UpdateTimeToLiveà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

Activation de la TTL (Durée de vie) sur une table DynamoDB existante à l'aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse enableTTL(final String tableName, final
 String attributeName, final Region region) {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(true)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();
```

```
        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

Désactivation de la TTL (Durée de vie) sur une table DynamoDB existante à l'aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse disableTTL(
        final String tableName, final String attributeName, final Region region)
 {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(false)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
```

```
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateTimeToLiveà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Activation de la TTL (Durée de vie) sur une table DynamoDB existante.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') =>
 {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
```

```
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Désactivation de la TTL (Durée de vie) sur une table DynamoDB existante.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1')
 => {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
```

```
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Pour plus de détails sur l'API, reportez-vous UpdateTimeToLiveà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Activation de la TTL (Durée de vie) sur une table DynamoDB existante.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
```

```
            TimeToLiveSpecification={"Enabled": True, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(
                f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}"
            )
        return response
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
enable_ttl("your-table-name", "expireAt")
```

Désactivation de la TTL (Durée de vie) sur une table DynamoDB existante.

```
import boto3


def disable_ttl(table_name, ttl_attribute_name):
    """
    Disables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table being modified
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
```

```
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": False, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been disabled successfully.")
        else:
            print(
                f"Failed to disable TTL, status code
 {response['ResponseMetadata']['HTTPStatusCode']}"
            )
    except Exception as ex:
        print("Couldn't disable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
disable_ttl("your-table-name", "expireAt")
```

- Pour plus de détails sur l'API, consultez UpdateTimeToLivele AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Scénarios pour DynamoDB utilisant AWS SDKs

Les exemples de code suivants vous montrent comment implémenter des scénarios courants dans DynamoDB avec.AWS SDKs Ces scénarios vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions dans DynamoDB ou en les combinant avec d'autres Services AWS. Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code.

Les scénarios ciblent un niveau d'expérience intermédiaire pour vous aider à comprendre les actions de service dans leur contexte.

Exemples

- Accélérez les lectures DynamoDB avec DAX à l'aide d'un SDK AWS
- Travaillez avec des scénarios d'index secondaire global DynamoDB avancés à l'aide de la version v2 AWS Command Line Interface
- Créer une application pour soumettre des données à une table DynamoDB
- Comparez plusieurs valeurs avec un seul attribut dans DynamoDB à l'aide d'un SDK AWS
- Mettre à jour de manière conditionnelle un élément DynamoDB avec un TTL à l'aide d'un SDK AWS
- Connectez-vous à une instance DynamoDB locale à l'aide d'un SDK AWS
- Compter les opérateurs d'expression dans DynamoDB à l'aide d'un SDK AWS
- Créer une API REST API Gateway pour suivre les données de la COVID-19
- Créer une application de messagerie avec Step Functions
- Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l’aide d’étiquettes
- Création d'une table DynamoDB avec un index secondaire global à l'aide du SDK AWS
- Création d'une table DynamoDB avec un paramètre de débit à chaud à l'aide d'un SDK AWS
- Créer une application web pour suivre les données DynamoDB
- Créer une application de chat Websocket avec API Gateway
- Création d'un élément DynamoDB avec un TTL à l'aide d'un SDK AWS
- Créez et gérez des tables globales DynamoDB avec une forte cohérence multirégionale à l'aide d'un SDK AWS
- Création et gestion de tables globales DynamoDB illustrant MREC à l'aide d'un SDK AWS
- Supprimer des données DynamoDB à l'aide des instructions PARTIQL DELETE avec un SDK AWS
- Détectez le PPE dans les images avec Amazon Rekognition à l'aide d'un SDK AWS
- Insérer des données DynamoDB à l'aide des instructions PartiQL INSERT avec un SDK AWS
- Invoquer une fonction Lambda à partir d’un navigateur
- Gestion des index secondaires globaux DynamoDB à l'aide de la version 2 AWS Command Line Interface
- Gérez les politiques basées sur les ressources DynamoDB à l'aide de la version 2 AWS Command Line Interface

- Surveillez les performances d'Amazon DynamoDB à l'aide d'un SDK AWS
- Exécuter des opérations de requête DynamoDB avancées à l'aide d'un SDK AWS
- Effectuez des opérations de liste dans DynamoDB à l'aide d'un SDK AWS
- Effectuez des opérations cartographiques dans DynamoDB à l'aide d'un SDK AWS
- Exécution d'opérations définies dans DynamoDB à l'aide d'un SDK AWS
- Interrogez une table DynamoDB à l'aide de lots d'instructions PartiQL et d'un SDK AWS
- Interroger une table DynamoDB à l'aide de PartiQL et d'un SDK AWS
- Interrogation d'une table DynamoDB à l'aide d'un index secondaire global avec un SDK AWS
- Interrogez une table DynamoDB à l'aide d'une condition begins_with avec un SDK AWS
- Interrogez une table DynamoDB en utilisant une plage de dates dans la clé de tri avec un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre dynamique à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec une expression de filtre et limitez avec un SDK AWS
- Interrogez une table DynamoDB avec des attributs imbriqués à l'aide d'un SDK AWS
- Interroger une table DynamoDB avec pagination à l'aide d'un SDK AWS
- Interrogez une table DynamoDB avec des lectures très cohérentes à l'aide d'un SDK AWS
- Interrogez les données DynamoDB à l'aide des instructions PartiQL SELECT avec un SDK AWS
- Interrogez une table DynamoDB pour les éléments TTL à l'aide d'un SDK AWS
- Interrogez les tables DynamoDB à l'aide de modèles de date et d'heure avec un SDK AWS
- Enregistrez les informations EXIF et autres images à l'aide d'un SDK AWS
- Configuration du contrôle d'accès basé sur les attributs pour DynamoDB à l'aide de la version 2 AWS Command Line Interface
- Comprendre l'ordre des expressions de mise à jour dans DynamoDB à l'aide d'un SDK AWS
- Mettre à jour un paramètre de table DynamoDB avec un débit chaud à l'aide d'un SDK AWS
- Mettre à jour un élément DynamoDB avec un TTL à l'aide d'un SDK AWS
- Mettre à jour les données DynamoDB à l'aide des instructions PARTIQL UPDATE avec un SDK AWS
- Utiliser API Gateway pour invoquer une fonction Lambda
- Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda
- Utiliser un modèle de document pour DynamoDB à l'aide d'un SDK AWS

- Utiliser un modèle de persistance des objets de haut niveau pour DynamoDB à l'aide d'un SDK AWS
- Utiliser les opérations de compteur atomique dans DynamoDB avec un SDK AWS
- Utiliser des opérations conditionnelles dans DynamoDB avec un SDK AWS
- Utiliser les noms d'attributs d'expression dans DynamoDB avec un SDK AWS
- Utilisent des événements planifiés pour invoquer une fonction Lambda
- Utilisation des index secondaires locaux DynamoDB à l'aide de la version v2 AWS Command Line Interface
- Utilisation de DynamoDB Streams et utilisation de la version 2 Time-to-Live AWS Command Line Interface
- Utilisez les tables globales DynamoDB et la réplication multirégionale avec une cohérence éventuelle (MREC) à l'aide de la version v2 AWS Command Line Interface
- Utiliser le balisage des ressources DynamoDB à l'aide de la version v2 AWS Command Line Interface
- Utiliser le chiffrement des tables DynamoDB à l'aide de la version v2 AWS Command Line Interface

Accélérez les lectures DynamoDB avec DAX à l'aide d'un SDK AWS

L'exemple de code suivant illustre comment :

- Créez et écrivez des données dans une table avec les clients DAX et SDK.
- Obtenez, interrogez et analysez la table avec les deux clients et comparez leurs performances.

Pour plus d'informations, consultez Développement avec le client DynamoDB Accelerator.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une table avec le client DAX ou Boto3.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

Écrivez les données de test dans la table.

```
import boto3
```

```
def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

Obtenez des éléments pour un certain nombre d’itérations pour le client DAX et le client Boto3, et indiquez le temps passé pour chacun d’eux.

```
import argparse
```

```
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
```

```
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

Interrogez la table pour un certain nombre d'itérations pour le client DAX et le client Boto3, et indiquez le temps passé pour chacune d'elles.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
```

```
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
```

```
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
client.")
        # Use a with statement so the DAX client closes the cluster after
completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Analysez la table à la recherche d'un certain nombre d'itérations pour le client DAX et le client Boto3, et indiquez le temps passé pour chacune d'elles.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
```

```
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Supprimez la table.

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Travaillez avec des scénarios d'index secondaire global DynamoDB avancés à l'aide de la version v2 AWS Command Line Interface

L'exemple de code suivant montre comment utiliser les configurations d'index secondaire global (GSI) avancées.

- Créez une table avec plusieurs GSIs.
- Créez une table avec une capacité à la demande et un GSI.
- Placez les éléments dans un tableau comportant plusieurs éléments GSIs.
- Interrogez plusieurs GSIs avec des conditions différentes.

Bash

AWS CLI avec le script Bash

Créez une table avec plusieurs GSIs.

```
# Create a table with multiple GSIs
aws dynamodb create-table \
    --table-name MusicLibrary \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
        AttributeName=Year,AttributeType=N \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumIndex\",
                \"KeySchema\": [{\"AttributeName\":\"AlbumTitle\",\"KeyType\":
\"HASH\"}],
```

```
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            },
            {
                \"IndexName\": \"GenreYearIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Year\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Artist\",\"SongTitle\"]}
            }
        ]"
```

Créez une table avec une capacité à la demande et un GSI.

```
# Create a table with on-demand capacity and GSI
aws dynamodb create-table \
    --table-name MusicOnDemand \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GenreIndex\",
                \"KeySchema\": [{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH
\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            }
        ]"
```

Placez les éléments dans un tableau comportant plusieurs éléments GSIs.

```
# Add items to MusicLibrary table
aws dynamodb put-item \
    --table-name MusicLibrary \
```

```
    --item '{
        "Artist": {"S": "The Beatles"},
        "SongTitle": {"S": "Hey Jude"},
        "AlbumTitle": {"S": "Past Masters"},
        "Genre": {"S": "Rock"},
        "Year": {"N": "1968"}
    }'

aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "Miles Davis"},
        "SongTitle": {"S": "So What"},
        "AlbumTitle": {"S": "Kind of Blue"},
        "Genre": {"S": "Jazz"},
        "Year": {"N": "1959"}
    }'
```

Interrogez les éléments d'une table contenant plusieurs éléments GSIs.

```
# Query the AlbumIndex GSI
echo "Querying AlbumIndex GSI:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Kind of Blue"}}'

# Query the GenreYearIndex GSI with a range condition
echo "Querying GenreYearIndex GSI with range condition:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name GenreYearIndex \
    --key-condition-expression "Genre = :genre AND #yr > :year" \
    --expression-attribute-names '{"#yr": "Year"}' \
    --expression-attribute-values '{":genre":{"S":"Rock"},":year":{"N":"1965"}}'
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable

- PutItem
- Interrogation

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créer une application pour soumettre des données à une table DynamoDB

Les exemples de code suivants montrent comment créer une application qui soumet des données à une table Amazon DynamoDB et vous avertit lorsqu'un utilisateur met à jour la table.

Java

SDK pour Java 2.x

Indique comment créer une application Web dynamique qui envoie des données à l'aide de l'API Java Amazon DynamoDB et envoie un message texte à l'aide de l'API Java Amazon Simple Notification Service.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SNS

JavaScript

SDK pour JavaScript (v3)

Cet exemple montre comment créer une application qui permet aux utilisateurs de soumettre des données à une table Amazon DynamoDB et d'envoyer un message texte à l'administrateur à l'aide d'Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SNS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Comparez plusieurs valeurs avec un seul attribut dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment comparer plusieurs valeurs avec un seul attribut dans DynamoDB.

- Utilisez l'opérateur IN pour comparer plusieurs valeurs avec un seul attribut.
- Comparez l'opérateur IN avec plusieurs conditions OR.
- Comprenez les avantages liés à l'utilisation d'IN en matière de performances et de complexité d'expression.

Java

SDK pour Java 2.x

Comparez plusieurs valeurs avec un seul attribut dans DynamoDB à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Locale;
import java.util.Map;

    /**
     * Queries a table using the IN operator to compare multiple values with a
 single attribute.
     *
     * <p>This method demonstrates how to use the IN operator in a filter
 expression
     * to match an attribute against multiple values.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareMultipleValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
```

```
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Queries a table using multiple OR conditions to compare multiple values
with a single attribute.
     *
     * <p>This method demonstrates the alternative approach to using the IN
operator,
     * by using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareWithOrConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
```

```
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for OR conditions
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the OR conditions
        StringBuilder orConditions = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                orConditions.append(" OR ");
            }
            orConditions.append("#attrName = :val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression(orConditions.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Compares the performance of using the IN operator versus multiple OR
conditions.
     *
     * <p>This method demonstrates the performance difference between using the
IN operator
```

```
     * and using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return Map containing the performance comparison results
     */
    public static Map<String, Object> comparePerformance(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Measure performance of IN operator
            long inStartTime = System.nanoTime();
            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long inEndTime = System.nanoTime();
            long inDuration = inEndTime - inStartTime;

            // Measure performance of OR conditions
            long orStartTime = System.nanoTime();
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long orEndTime = System.nanoTime();
            long orDuration = orEndTime - orStartTime;

            // Record results
            results.put("inOperatorDuration", inDuration);
            results.put("orConditionsDuration", orDuration);
            results.put("inOperatorItems", inResponse.count());
            results.put("orConditionsItems", orResponse.count());
            results.put("inOperatorExpression", "IN operator with " +
valuesList.size() + " values");
```

```
            results.put("orConditionsExpression", valuesList.size() + " OR
conditions");
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Scans a table using the IN operator with a large number of values.
     *
     * <p>This method demonstrates how to use the IN operator with a large number
of values,
     * which can help stay within the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The scan response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithLargeInClause(
        DynamoDbClient dynamoDbClient, String tableName, String attributeName,
List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
```

```
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }

    /**
     * Generates a list of sample values for testing.
     *
     * <p>Helper method to generate a list of sample values for testing.
     *
     * @param valueType The type of values to generate (string, number, or
boolean)
     * @param count The number of values to generate
     * @return List of generated attribute values
     */
    public static List<AttributeValue> generateSampleValues(String valueType, int
count) {
        List<AttributeValue> values = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            AttributeValue value;

            switch (valueType.toLowerCase(Locale.ROOT)) {
                case "string":
                    value = AttributeValue.builder().s("Value" + i).build();
                    break;
                case "number":
                    value =
AttributeValue.builder().n(String.valueOf(i)).build();
                    break;
```

```
                case "boolean":
                    value = AttributeValue.builder().bool(i % 2 == 0).build();
                    break;
                default:
                    throw new IllegalArgumentException("Unsupported value type: "
+ valueType);
            }

            values.add(value);
        }

        return values;
    }
```

Exemple d'utilisation de la comparaison de plusieurs valeurs avec AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        System.out.println("Demonstrating how to compare multiple values with a
single attribute in DynamoDB");

        try {
            // Example 1: Using the IN operator
            System.out.println("\nExample 1: Using the IN operator");
            List<AttributeValue> categories = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Computers").build(),
                AttributeValue.builder().s("Accessories").build());

            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + inResponse.count() + " items using IN
operator");
            System.out.println("Items: " + inResponse.items());

            // Example 2: Using multiple OR conditions
```

```
            System.out.println("\nExample 2: Using multiple OR conditions");
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + orResponse.count() + " items using OR
conditions");
            System.out.println("Items: " + orResponse.items());

            // Example 3: Performance comparison
            System.out.println("\nExample 3: Performance comparison");
            Map<String, Object> perfComparison = comparePerformance(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            if ((boolean) perfComparison.get("success")) {
                System.out.println("IN operator duration: " +
perfComparison.get("inOperatorDuration") + " ns");
                System.out.println("OR conditions duration: " +
perfComparison.get("orConditionsDuration") + " ns");
                System.out.println("IN operator found " +
perfComparison.get("inOperatorItems") + " items");
                System.out.println("OR conditions found " +
perfComparison.get("orConditionsItems") + " items");
                System.out.println("Expression complexity comparison:");
                System.out.println("  IN operator: " +
perfComparison.get("inOperatorExpression"));
                System.out.println("  OR conditions: " +
perfComparison.get("orConditionsExpression"));
            } else {
                System.out.println("Performance comparison failed: " +
perfComparison.get("error"));
            }

            // Example 4: Using IN with a large number of values
```

```
            System.out.println("\nExample 4: Using IN with a large number of
values");
            List<AttributeValue> productIds = generateSampleValues("string", 20);

            ScanResponse largeInResponse = scanWithLargeInClause(dynamoDbClient,
tableName, "ProductId", productIds);

            System.out.println(
                "Found " + largeInResponse.count() + " items using IN with " +
productIds.size() + " values");

            // Explain the benefits of using IN
            System.out.println("\nKey points about using the IN operator in
DynamoDB:");
            System.out.println("1. The IN operator allows comparing a single
attribute against multiple values");
            System.out.println("2. IN is more concise than using multiple OR
conditions");
            System.out.println("3. IN counts as only 1 operator regardless of the
number of values");
            System.out.println("4. Multiple OR conditions count as 1 operator per
condition plus 1 per OR");
            System.out.println("5. Using IN helps stay within the 300 operator
limit for complex expressions");
            System.out.println("6. IN can be used in filter expressions and
condition expressions");
            System.out.println("7. The IN operator supports up to 100 comparison
values");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x.
  - Interrogation
  - Analyser

JavaScript

SDK pour JavaScript (v3)

Comparez plusieurs valeurs avec un seul attribut en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple
 OR conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
 const docClient = DynamoDBDocumentClient.from(client);

 // Create the filter expression using the IN operator
 const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
`:val${index}`).join(', ')})`;

 // Create expression attribute values for the values list
 const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
   acc[`:val${index}`] = val;
   return acc;
 }, {});

 // If partition key is provided, perform a query operation
 if (partitionKeyName && partitionKeyValue) {
   const keyCondition = `${partitionKeyName} = :partitionKey`;
   expressionAttributeValues[':partitionKey'] = partitionKeyValue;

   // Initialize array to collect all items
   let allItems = [];
   let lastEvaluatedKey;

   // Use pagination to get all results
   do {
     const params = {
       TableName: tableName,
       KeyConditionExpression: keyCondition,
       FilterExpression: filterExpression,
       ExpressionAttributeValues: expressionAttributeValues
     };

     // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
     if (lastEvaluatedKey) {
       params.ExclusiveStartKey = lastEvaluatedKey;
     }

     const response = await docClient.send(new QueryCommand(params));

     // Add the items from this page to our collection
     if (response.Items && response.Items.length > 0) {
       allItems = [...allItems, ...response.Items];
     }

     // Get the key for the next page of results
```

```
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
```

```
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
```

```
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);
```

```
    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Exemple d'utilisation de la comparaison de plusieurs valeurs avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];

  console.log(`Searching for products in any of these categories:
 ${valuesList.join(', ')}`);

  try {
    // Using the IN operator (recommended approach)
    console.log("\nApproach 1: Using the IN operator");
    const response = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response.Count} products in the specified categories`);

    // Using multiple OR conditions (alternative approach)
    console.log("\nApproach 2: Using multiple OR conditions");
    const response2 = await compareWithOrConditions(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response2.Count} products in the specified categories`);

    // Example with a query operation
    console.log("\nQuerying a specific manufacturer's products in multiple
 categories");
```

```
    const partitionKeyName = "Manufacturer";
    const partitionKeyValue = "Acme";

    const response3 = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList,
      partitionKeyName,
      partitionKeyValue
    );

    console.log(`Found ${response3.Count} Acme products in the specified
 categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript.
  - Interrogation
  - Analyser

Python

Kit SDK for Python (Boto3)

Comparez plusieurs valeurs avec un seul attribut en utilisant AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key
```

```
from typing import Any, Dict, List, Optional


def compare_multiple_values(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.

    This function demonstrates the use of the IN operator to compare a single
 attribute
    against multiple possible values, which is more efficient than using multiple
 OR conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create the filter expression using the is_in method
    filter_expression = Attr(attribute_name).is_in(values_list)

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
```

```
            KeyConditionExpression=key_condition,
 FilterExpression=filter_expression
        )
    else:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}


def compare_with_or_conditions(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Alternative implementation using multiple OR conditions instead of the IN
 operator.

    This function is provided for comparison to show why using the IN operator is
 preferable.
    With many values, this approach becomes verbose and less efficient.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        attribute_name (str): The name of the attribute to compare against the
values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create a filter expression with multiple OR conditions
    filter_expression = None
    for value in values_list:
        condition = Attr(attribute_name).eq(value)
        if filter_expression is None:
            filter_expression = condition
        else:
            filter_expression = filter_expression | condition

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value and filter_expression:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
FilterExpression=filter_expression
        )
    elif filter_expression:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)
    else:
        # Return empty response if no values provided
        return {"Items": [], "Count": 0}

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
```

```
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}
```

Exemple d'utilisation de la comparaison de plusieurs valeurs avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the compare_multiple_values function."""
    # Example parameters
    table_name = "Products"
    attribute_name = "Category"
    values_list = ["Electronics", "Computers", "Accessories"]

    print(f"Searching for products in any of these categories: {values_list}")

    # Using the IN operator (recommended approach)
    print("\nApproach 1: Using the IN operator")
    response = compare_multiple_values(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )

    print(f"Found {response['Count']} products in the specified categories")

    # Using multiple OR conditions (alternative approach)
    print("\nApproach 2: Using multiple OR conditions")
    response2 = compare_with_or_conditions(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )
```

```
    print(f"Found {response2['Count']} products in the specified categories")

    # Example with a query operation
    print("\nQuerying a specific manufacturer's products in multiple categories")
    partition_key_name = "Manufacturer"
    partition_key_value = "Acme"

    response3 = compare_multiple_values(
        table_name=table_name,
        attribute_name=attribute_name,
        values_list=values_list,
        partition_key_name=partition_key_name,
        partition_key_value=partition_key_value,
    )

    print(f"Found {response3['Count']} Acme products in the specified
categories")

    # Explain the benefits of using the IN operator
    print("\nBenefits of using the IN operator:")
    print("1. More concise expression compared to multiple OR conditions")
    print("2. Better readability and maintainability")
    print("3. Potentially better performance with large value lists")
    print("4. Simpler code that's less prone to errors")
    print("5. Easier to modify when adding or removing values")
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Python (Boto3).
  - Interrogation
  - Analyser

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Mettre à jour de manière conditionnelle un élément DynamoDB avec un TTL à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment mettre à jour le TTL d'un élément de manière conditionnelle.

Java

SDK pour Java 2.x

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
```

```
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
or partition key.
                sortKey - The name of the sort key. Also known as the range
attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
is located in. (Default: us-east-1)
            """;
   private static final int DAYS_TO_EXPIRE = 90;
   private static final int SECONDS_PER_DAY = 24 * 60 * 60;
   private static final String PRIMARY_KEY_ATTR = "primaryKey";
   private static final String SORT_KEY_ATTR = "sortKey";
   private static final String UPDATED_AT_ATTR = "updatedAt";
   private static final String EXPIRE_AT_ATTR = "expireAt";
   private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
   private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
   private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
   private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
   private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

   private final DynamoDbClient dynamoDbClient;

   /**
    * Constructs an UpdateTTLConditional with a default DynamoDB client.
    */
   public UpdateTTLConditional() {
       this.dynamoDbClient = null;
   }

   /**
    * Constructs an UpdateTTLConditional with the specified DynamoDB client.
    *
    * @param dynamoDbClient The DynamoDB client to use
    */
   public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
       this.dynamoDbClient = dynamoDbClient;
   }

   /**
```

```
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
```

```
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
```

```
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)
```

```
        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Connectez-vous à une instance DynamoDB locale à l'aide d'un SDK AWS

L'exemple de code suivant montre comment remplacer l'URL d'un point de terminaison pour se connecter à un déploiement de développement local de DynamoDB et à un SDK.AWS

Pour plus d'informations, consultez DynamoDB local.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
    let dynamodb_local_config =
 aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);
```

```
    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Compter les opérateurs d'expression dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment compter les opérateurs d’expression dans DynamoDB.

- Compréhension de la limite de 300 opérateurs de DynamoDB.
- Comptage des opérateurs dans les expressions complexes.
- Optimisation des expressions pour respecter les limites.

Java

SDK pour Java 2.x

Démonstration du comptage des opérateurs d’expression à l’aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

    /**
     * Creates a complex filter expression with a specified number of conditions.
     *
     * <p>This method demonstrates how to generate a complex expression with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param conditionsCount Number of conditions to include
     * @param useAnd Whether to use AND (true) or OR (false) between conditions
     * @return Map containing the filter expression, attribute values, and
 operator count
     */
    public static Map<String, Object> createComplexFilterExpression(int
 conditionsCount, boolean useAnd) {
        // Initialize the expression parts and attribute values
        StringBuilder filterExpression = new StringBuilder();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of conditions
        for (int i = 0; i < conditionsCount; i++) {
            // Add the operator between conditions (except for the first one)
            if (i > 0) {
                filterExpression.append(useAnd ? " AND " : " OR ");
            }

            // Alternate between different comparison operators for variety
            String valueKey = ":val" + i;

            switch (i % 5) {
                case 0:
                    filterExpression.append("attribute").append(i).append(" =
 ").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
 i).build());
                    break;
                case 1:
```

```
                    filterExpression.append("attribute").append(i).append(" >
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    filterExpression.append("attribute").append(i).append(" <
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                case 3:
                    filterExpression
                        .append("contains(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("substring" +
i).build());
                    break;
                case 4:
                    filterExpression
                        .append("attribute_exists(attribute")
                        .append(i)
                        .append(")");
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each condition has 1 operator (=, >, <, contains, attribute_exists)
        // Each AND or OR between conditions is 1 operator
        int operatorCount = conditionsCount + (conditionsCount > 0 ?
conditionsCount - 1 : 0);
```

```
        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("filterExpression", filterExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Creates a complex update expression with a specified number of operations.
     *
     * <p>This method demonstrates how to generate a complex update expression
with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param operationsCount Number of operations to include
     * @return Map containing the update expression, attribute values, and
operator count
     */
    public static Map<String, Object> createComplexUpdateExpression(int
operationsCount) {
        // Initialize the expression parts and attribute values
        StringBuilder updateExpression = new StringBuilder("SET ");
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of SET operations
        for (int i = 0; i < operationsCount; i++) {
            // Add comma between operations (except for the first one)
            if (i > 0) {
                updateExpression.append(", ");
            }

            // Alternate between different types of SET operations
            String valueKey = ":val" + i;

            switch (i % 3) {
                case 0:
                    // Simple assignment (1 operator: =)
                    updateExpression.append("attribute").append(i).append(" =
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
i).build());
```

```
                    break;
                case 1:
                    // Addition (2 operators: = and +)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = attribute")
                        .append(i)
                        .append(" + ")
                        .append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    // Conditional assignment with if_not_exists (2 operators: =
and if_not_exists)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = if_not_exists(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each operation has 1-2 operators as noted above
        int operatorCount = 0;
        for (int i = 0; i < operationsCount; i++) {
            operatorCount += (i % 3 == 0) ? 1 : 2;
        }
```

```
        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("updateExpression", updateExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Test the operator limit by attempting an operation with a complex
expression.
     *
     * <p>This method demonstrates what happens when an expression approaches or
     * exceeds the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param operatorCount Target number of operators to include
     * @return Map containing the result of the operation attempt
     */
    public static Map<String, Object> testOperatorLimit(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int operatorCount) {

        // Create a complex update expression with the specified operator count
        Map<String, Object> expressionData =
            createComplexUpdateExpression((int) Math.ceil(operatorCount /
1.5)); // Adjust to get close to target count

        String updateExpression = (String)
expressionData.get("updateExpression");
        @SuppressWarnings("unchecked")
        Map<String, AttributeValue> expressionAttributeValues =
            (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
        int actualCount = (int) expressionData.get("operatorCount");

        System.out.println("Generated update expression with approximately " +
actualCount + " operators");

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
```

```
            .tableName(tableName)
            .key(key)
            .updateExpression(updateExpression)
            .expressionAttributeValues(expressionAttributeValues)
            .returnValues("UPDATED_NEW")
            .build();

        try {
            // Attempt the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            Map<String, Object> result = new HashMap<>();
            result.put("success", true);
            result.put("message", "Operation succeeded with " + actualCount + "
operators");
            result.put("data", response);
            return result;

        } catch (DynamoDbException e) {
            // Check if the error is due to exceeding the operator limit
            if (e.getMessage().contains("too many operators")) {
                Map<String, Object> result = new HashMap<>();
                result.put("success", false);
                result.put("message", "Operation failed: " + e.getMessage());
                result.put("operatorCount", actualCount);
                return result;
            }

            // Return other errors
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "Operation failed: " + e.getMessage());
            result.put("error", e);
            return result;
        }
    }

    /**
     * Break down a complex expression into multiple simpler operations.
     *
     * <p>This method demonstrates how to handle expressions that would exceed
     * the 300 operator limit by breaking them into multiple operations.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param totalOperations Total number of operations to perform
     * @return Map containing the results of the operations
     */
    public static Map<String, Object> breakDownComplexExpression(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int totalOperations) {

        // Calculate how many operations we can safely include in each batch
        // Using 150 as a conservative limit (well below 300)
        final int operationsPerBatch = 100;
        final int batchCount = (int) Math.ceil((double) totalOperations /
operationsPerBatch);

        System.out.println("Breaking down " + totalOperations + " operations into
" + batchCount + " batches");

        Map<String, Object> results = new HashMap<>();
        results.put("totalBatches", batchCount);

        Map<Integer, Map<String, Object>> batchResults = new HashMap<>();

        // Process each batch
        for (int batch = 0; batch < batchCount; batch++) {
            // Calculate the operations for this batch
            int batchStart = batch * operationsPerBatch;
            int batchEnd = Math.min(batchStart + operationsPerBatch,
totalOperations);
            int batchSize = batchEnd - batchStart;

            System.out.println(
                "Processing batch " + (batch + 1) + "/" + batchCount + " with " +
batchSize + " operations");

            // Create an update expression for this batch
            Map<String, Object> expressionData =
createComplexUpdateExpression(batchSize);

            String updateExpression = (String)
expressionData.get("updateExpression");
            @SuppressWarnings("unchecked")
            Map<String, AttributeValue> expressionAttributeValues =
```

```
                (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
            int operatorCount = (int) expressionData.get("operatorCount");

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression(updateExpression)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues("UPDATED_NEW")
                .build();

            try {
                // Perform the update operation for this batch
                UpdateItemResponse response = dynamoDbClient.updateItem(request);

                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", true);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("attributes", response.attributes());

                batchResults.put(batch, batchResult);

            } catch (DynamoDbException e) {
                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", false);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("error", e.getMessage());

                batchResults.put(batch, batchResult);

                // Continue with next batch instead of breaking
                continue;
            }
        }

        results.put("results", batchResults);
        return results;
    }

    /**
```

```
     * Count operators in a DynamoDB expression based on the rules in the
 documentation.
     *
     * <p>This method demonstrates how operators are counted according to the
     * DynamoDB documentation.
     *
     * @param expression The DynamoDB expression to analyze
     * @return Map containing the breakdown of operator counts
     */
    public static Map<String, Integer> countOperatorsInExpression(String
 expression) {
        // Initialize counters for different operator types
        Map<String, Integer> counts = new HashMap<>();
        counts.put("comparisonOperators", 0);
        counts.put("logicalOperators", 0);
        counts.put("functions", 0);
        counts.put("arithmeticOperators", 0);
        counts.put("specialOperators", 0);
        counts.put("total", 0);

        // Count comparison operators (=, <>, <, <=, >, >=)
        // This is a simplified approach and may not catch all cases
        int comparisonCount = 0;
        Pattern comparisonPattern = Pattern.compile("(=|<>|<=|>=|<|>)");
        Matcher comparisonMatcher = comparisonPattern.matcher(expression);
        while (comparisonMatcher.find()) {
            comparisonCount++;
        }
        counts.put("comparisonOperators", comparisonCount);

        // Count logical operators (AND, OR, NOT)
        int andCount = countOccurrences(expression, "\\bAND\\b");
        int orCount = countOccurrences(expression, "\\bOR\\b");
        int notCount = countOccurrences(expression, "\\bNOT\\b");
        counts.put("logicalOperators", andCount + orCount + notCount);

        // Count functions (attribute_exists, attribute_not_exists,
 attribute_type, begins_with, contains, size)
        int functionCount = countOccurrences(
            expression,
            "\\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\\(");
        counts.put("functions", functionCount);
```

```
        // Count arithmetic operators (+ and -)
        // This is a simplified approach and may not catch all cases
        int arithmeticCount = 0;
        Pattern arithmeticPattern = Pattern.compile("[a-zA-Z0-9_)\\]]\\s*[\\+\
\-]\\s*[a-zA-Z0-9_:(]");
        Matcher arithmeticMatcher = arithmeticPattern.matcher(expression);
        while (arithmeticMatcher.find()) {
            arithmeticCount++;
        }
        counts.put("arithmeticOperators", arithmeticCount);

        // Count special operators (BETWEEN, IN)
        int betweenCount = countOccurrences(expression, "\\bBETWEEN\\b");
        int inCount = countOccurrences(expression, "\\bIN\\b");
        counts.put("specialOperators", betweenCount + inCount);

        // Add extra operators for BETWEEN (each BETWEEN includes an AND)
        int currentLogicalOps = counts.getOrDefault("logicalOperators", 0);
        counts.put("logicalOperators", currentLogicalOps + betweenCount);

        // Calculate total
        int total = counts.getOrDefault("comparisonOperators", 0)
            + counts.getOrDefault("logicalOperators", 0)
            + counts.getOrDefault("functions", 0)
            + counts.getOrDefault("arithmeticOperators", 0)
            + counts.getOrDefault("specialOperators", 0);
        counts.put("total", total);

        return counts;
    }

    /**
     * Helper method to count occurrences of a pattern in a string.
     *
     * @param text The text to search in
     * @param regex The regular expression pattern to search for
     * @return The number of occurrences
     */
    private static int countOccurrences(String text, String regex) {
        final Pattern pattern = Pattern.compile(regex);
        final Matcher matcher = pattern.matcher(text);
        int count = 0;
        while (matcher.find()) {
            count++;
```

```
        }
        return count;
    }
```

Exemple d'utilisation de l'opérateur d'expression comptant avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating DynamoDB expression operator counting
and the 300 operator limit");

        try {
            // Example 1: Analyze a simple expression
            System.out.println("\nExample 1: Analyzing a simple expression");
            String simpleExpression = "Price = :price AND Rating > :rating AND
Category IN (:cat1, :cat2, :cat3)";
            Map<String, Integer> simpleCount =
countOperatorsInExpression(simpleExpression);

            System.out.println("Expression: " + simpleExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
simpleCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
simpleCount.get("logicalOperators"));
            System.out.println("- Functions: " + simpleCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
simpleCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
simpleCount.get("specialOperators"));
            System.out.println("- Total operators: " + simpleCount.get("total"));

            // Example 2: Analyze a complex expression
            System.out.println("\nExample 2: Analyzing a complex expression");
            String complexExpression = "(attribute_exists(Category) AND Size
BETWEEN :min AND :max) OR "
                + "(Price > :price AND contains(Description, :keyword) AND "
                + "(Rating >= :minRating OR Reviews > :minReviews))";
```

```
            Map<String, Integer> complexCount =
countOperatorsInExpression(complexExpression);

            System.out.println("Expression: " + complexExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
complexCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
complexCount.get("logicalOperators"));
            System.out.println("- Functions: " + complexCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
complexCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
complexCount.get("specialOperators"));
            System.out.println("- Total operators: " +
complexCount.get("total"));

            // Example 3: Test approaching the operator limit
            System.out.println("\nExample 3: Testing an expression approaching
the operator limit");
            Map<String, Object> approachingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 290);
            System.out.println(approachingLimit.get("message"));

            // Example 4: Test exceeding the operator limit
            System.out.println("\nExample 4: Testing an expression exceeding the
operator limit");
            Map<String, Object> exceedingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 310);
            System.out.println(exceedingLimit.get("message"));

            // Example 5: Breaking down a complex expression
            System.out.println("\nExample 5: Breaking down a complex expression
into multiple operations");
            Map<String, Object> breakdownResult =
breakDownComplexExpression(dynamoDbClient, tableName, key, 500);
            @SuppressWarnings("unchecked")
            Map<Integer, Map<String, Object>> results =
                (Map<Integer, Map<String, Object>>)
breakdownResult.get("results");
            System.out.println(
                "Processed " + results.size() + " of " +
breakdownResult.get("totalBatches") + " batches");
```

```
            // Explain the operator counting rules
            System.out.println("\nKey points about DynamoDB expression operator
counting:");
            System.out.println("1. The maximum number of operators in any
expression is 300");
            System.out.println("2. Each comparison operator (=, <>, <, <=, >, >=)
counts as 1 operator");
            System.out.println("3. Each logical operator (AND, OR, NOT) counts as
1 operator");
            System.out.println("4. Each function call (attribute_exists,
contains, etc.) counts as 1 operator");
            System.out.println("5. Each arithmetic operator (+ or -) counts as 1
operator");
            System.out.println("6. BETWEEN counts as 2 operators (BETWEEN itself
and the AND within it)");
            System.out.println("7. IN counts as 1 operator regardless of the
number of values");
            System.out.println("8. Parentheses for grouping and attribute paths
don't count as operators");
            System.out.println("9. When you exceed the limit, the error always
reports '301 operators'");
            System.out.println("10. For complex operations, break them into
multiple smaller operations");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez le comptage des opérateurs d’expression à l’aide du kit AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute
 values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
```

```
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount
 - 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute
 values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
```

```
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
```

```
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount:
 actualCount } =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to
 get close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
```

```
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
```

```
 // Using 150 as a conservative limit (well below 300)
 const operationsPerBatch = 100;
 const batchCount = Math.ceil(totalOperations / operationsPerBatch);

 console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

 const results = [];

 // Process each batch
 for (let batch = 0; batch < batchCount; batch++) {
   // Calculate the operations for this batch
   const batchStart = batch * operationsPerBatch;
   const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
   const batchSize = batchEnd - batchStart;

   console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

   // Create an update expression for this batch
   const { updateExpression, expressionAttributeValues, operatorCount } =
     createComplexUpdateExpression(batchSize);

   // Define the update parameters
   const params = {
     TableName: tableName,
     Key: key,
     UpdateExpression: updateExpression,
     ExpressionAttributeValues: expressionAttributeValues,
     ReturnValues: "UPDATED_NEW"
   };

   try {
     // Perform the update operation for this batch
     const response = await docClient.send(new UpdateCommand(params));

     results.push({
       batch: batch + 1,
       success: true,
       operatorCount,
       attributes: response.Attributes
     });
   } catch (error) {
     results.push({
```

```
        batch: batch + 1,
        success: false,
        operatorCount,
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the
 documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
```

```
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Exemple d'utilisation de l'opérateur d'expression comptant avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
```

```
 // Example parameters
 const config = { region: "us-west-2" };
 const tableName = "Products";
 const key = { ProductId: "P12345" };

 console.log("Demonstrating DynamoDB expression operator counting and the 300
operator limit");

 try {
   // Example 1: Analyze a simple expression
   console.log("\nExample 1: Analyzing a simple expression");
   const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
   const simpleCount = countOperatorsInExpression(simpleExpression);

   console.log(`Expression: ${simpleExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
   console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
   console.log(`- Functions: ${simpleCount.functions}`);
   console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
   console.log(`- Special operators: ${simpleCount.specialOperators}`);
   console.log(`- Total operators: ${simpleCount.total}`);

   // Example 2: Analyze a complex expression
   console.log("\nExample 2: Analyzing a complex expression");
   const complexExpression =
     "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
     "(Price > :price AND contains(Description, :keyword) AND " +
     "(Rating >= :minRating OR Reviews > :minReviews))";
   const complexCount = countOperatorsInExpression(complexExpression);

   console.log(`Expression: ${complexExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
   console.log(`- Logical operators: ${complexCount.logicalOperators}`);
   console.log(`- Functions: ${complexCount.functions}`);
   console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
   console.log(`- Special operators: ${complexCount.specialOperators}`);
   console.log(`- Total operators: ${complexCount.total}`);

   // Example 3: Test approaching the operator limit
   console.log("\nExample 3: Testing an expression approaching the operator
limit");
```

```
    const approachingLimit = await testOperatorLimit(config, tableName, key,
290);
    console.log(approachingLimit.message);

    // Example 4: Test exceeding the operator limit
    console.log("\nExample 4: Testing an expression exceeding the operator
limit");
    const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName,
key, 500);
    console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts
as 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND
within it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
operators");
    console.log("9. When you exceed the limit, the error always reports '301
operators'");
    console.log("10. For complex operations, break them into multiple smaller
operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez le comptage des opérateurs d'expression à l'aide du kit AWS SDK pour Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List, Optional, Tuple



def create_complex_filter_expression(
    attribute_name: str, values: List[Any], use_or: bool = True
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a complex filter expression with multiple conditions.

    This function demonstrates how to build a complex filter expression
    and count the number of operators used.

    Args:
        attribute_name (str): The name of the attribute to filter on.
        values (List[Any]): List of values to compare against.
        use_or (bool, optional): Whether to use OR between conditions. Defaults
 to True.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if not values:
        return "", {}, {}, 0

    # Initialize expression components
    filter_expression = ""
```

```
    expression_attribute_values = {}
    expression_attribute_names = {"#attr": attribute_name}
    operator_count = 0

    # Build the filter expression
    for i, value in enumerate(values):
        value_placeholder = f":val{i}"
        expression_attribute_values[value_placeholder] = value

        if i > 0:
            # Add OR or AND operator between conditions
            filter_expression += " OR " if use_or else " AND "
            operator_count += 1  # Count the OR/AND operator

        # Add the condition
        filter_expression += f"#attr = {value_placeholder}"
        operator_count += 1  # Count the = operator

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def create_nested_filter_expression(
    depth: int, conditions_per_level: int
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a deeply nested filter expression with multiple conditions.

    This function demonstrates how to build a complex nested filter expression
    and count the number of operators used.

    Args:
        depth (int): The depth of nesting.
        conditions_per_level (int): Number of conditions at each level.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
```

```
            - The number of operators used
    """
    if depth <= 0 or conditions_per_level <= 0:
        return "", {}, {}, 0

    # Initialize expression components
    expression_attribute_values = {}
    expression_attribute_names = {}
    operator_count = 0

    def build_nested_expression(current_depth: int, prefix: str) -> str:
        nonlocal operator_count

        if current_depth <= 0:
            return ""

        # Build conditions at this level
        conditions = []
        for i in range(conditions_per_level):
            attr_name = f"attr{prefix}_{i}"
            attr_placeholder = f"#attr{prefix}_{i}"
            val_placeholder = f":val{prefix}_{i}"

            expression_attribute_names[attr_placeholder] = attr_name
            expression_attribute_values[val_placeholder] = i

            conditions.append(f"{attr_placeholder} = {val_placeholder}")
            operator_count += 1  # Count the = operator

        # Join conditions with AND
        level_expression = " AND ".join(conditions)
        operator_count += max(0, len(conditions) - 1)  # Count the AND operators

        # If not at the deepest level, add nested expressions
        if current_depth > 1:
            nested_expr = build_nested_expression(current_depth - 1,
f"{prefix}_{current_depth}")
            if nested_expr:
                level_expression = f"({level_expression}) OR ({nested_expr})"
                operator_count += 1  # Count the OR operator

        return level_expression

    # Build the expression starting from the top level
```

```
    filter_expression = build_nested_expression(depth, "1")

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def count_operators_in_update_expression(update_expression: str) -> int:
    """
    Count the number of operators in an update expression.

    This function demonstrates how to count operators in an update expression
    based on DynamoDB's rules.

    Args:
        update_expression (str): The update expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count SET operations
    if "SET" in update_expression:
        set_section = (
            update_expression.split("SET")[1].split("REMOVE")[0].split("ADD")
[0].split("DELETE")[0]
        )

        # Count assignment operators (=)
        operator_count += set_section.count("=")

        # Count arithmetic operators (+, -)
        operator_count += set_section.count("+")
        operator_count += set_section.count("-")

        # Count list_append function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("list_append")

        # Count if_not_exists function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("if_not_exists")
```

```
    # Count REMOVE operations (no additional operators)

    # Count ADD operations (each ADD counts as 1 operator)
    if "ADD" in update_expression:
        add_section = (
            update_expression.split("ADD")[1].split("DELETE")[0].split("SET")
[0].split("REMOVE")[0]
        )
        operator_count += add_section.count(",") + 1

    # Count DELETE operations (each DELETE counts as 1 operator)
    if "DELETE" in update_expression:
        delete_section = (
            update_expression.split("DELETE")[1].split("SET")[0].split("ADD")
[0].split("REMOVE")[0]
        )
        operator_count += delete_section.count(",") + 1

    return operator_count


def count_operators_in_condition_expression(condition_expression: str) -> int:
    """
    Count the number of operators in a condition expression.

    This function demonstrates how to count operators in a condition expression
    based on DynamoDB's rules.

    Args:
        condition_expression (str): The condition expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count comparison operators
    comparison_operators = ["=", "<>", "<", "<=", ">", ">="]
    for op in comparison_operators:
        operator_count += condition_expression.count(op)

    # Count logical operators
    operator_count += condition_expression.upper().count(" AND ")
```

```
    operator_count += condition_expression.upper().count(" OR ")
    operator_count += condition_expression.upper().count("NOT ")

    # Count BETWEEN operator (counts as 2: BETWEEN + AND)
    between_count = condition_expression.upper().count(" BETWEEN ")
    operator_count += between_count * 2

    # Count IN operator (counts as 1 regardless of number of values)
    operator_count += condition_expression.upper().count(" IN ")

    # Count functions (each counts as 1 operator)
    functions = [
        "attribute_exists",
        "attribute_not_exists",
        "attribute_type",
        "begins_with",
        "contains",
        "size",
    ]
    for func in functions:
        operator_count += condition_expression.lower().count(func)

    return operator_count


# Note: This function is for demonstration purposes only and should be called
 from example_usage()
# It's not meant to be used directly as a test function
def _test_expression_limit(
    table_name: str, key: Dict[str, Any], operator_count: int, attribute_name:
 str = "TestAttribute"
) -> Tuple[bool, Optional[str]]:
    """
    Test if an expression with a specific number of operators exceeds the limit.

    This function demonstrates how to test the 300 operator limit by creating
    an expression with a specified number of operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        operator_count (int): The number of operators to include in the
 expression.
```

```
        attribute_name (str, optional): The name of the attribute to update.
Defaults to "TestAttribute".

    Returns:
        Tuple[bool, Optional[str]]: A tuple containing:
            - A boolean indicating if the operation succeeded
            - The error message if it failed, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create an update expression with the specified number of operators
    update_expression = f"SET #{attribute_name} = :val0"
    expression_attribute_names = {f"#{attribute_name}": attribute_name}
    expression_attribute_values = {":val0": 0}

    # Add additional SET operations to reach the desired operator count
    # Each assignment adds 1 operator
    for i in range(1, operator_count):
        attr_name = f"{attribute_name}{i}"
        attr_placeholder = f"#attr{i}"
        val_placeholder = f":val{i}"

        update_expression += f", {attr_placeholder} = {val_placeholder}"
        expression_attribute_names[attr_placeholder] = attr_name
        expression_attribute_values[val_placeholder] = i

    try:
        # Attempt the update operation
        table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values,
        )
        return True, None
    except ClientError as e:
        error_message = e.response["Error"]["Message"]

        if "expression contains too many operators" in error_message.lower():
            return False, error_message
        else:
            # Other error occurred
```

```
            raise
```

Exemple d'utilisation de l'opérateur d'expression comptant avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the expression operator counting functions."""

    print("Example 1: Creating a complex filter expression with multiple
 conditions")
    attribute_name = "Status"
    values = ["Active", "Pending", "Processing", "Shipped", "Delivered"]

    filter_expr, expr_attr_vals, expr_attr_names, op_count =
 create_complex_filter_expression(
        attribute_name=attribute_name, values=values, use_or=True
    )

    print(f"Filter Expression: {filter_expr}")
    print(f"Expression Attribute Values: {expr_attr_vals}")
    print(f"Expression Attribute Names: {expr_attr_names}")
    print(f"Operator Count: {op_count}")

    print("\nExample 2: Creating a nested filter expression")
    nested_expr, nested_vals, nested_names, nested_count =
 create_nested_filter_expression(
        depth=3, conditions_per_level=2
    )

    print(f"Nested Filter Expression: {nested_expr}")
    print(f"Operator Count: {nested_count}")

    print("\nExample 3: Counting operators in an update expression")
    update_expression = "SET #name = :name, #age = :age + :increment,
 #address.#city = :city, #status = if_not_exists(#status, :default_status) REMOVE
 #old_field ADD #counter :value DELETE #set_attr :set_val"
    update_op_count = count_operators_in_update_expression(update_expression)

    print(f"Update Expression: {update_expression}")
    print(f"Operator Count: {update_op_count}")
```

```
   print("\nExample 4: Counting operators in a condition expression")
   condition_expression = "(#status = :active OR #status = :pending) AND #price
BETWEEN :min_price AND :max_price AND attribute_exists(#category) AND NOT
(#stock <= :min_stock)"
   condition_op_count =
count_operators_in_condition_expression(condition_expression)

   print(f"Condition Expression: {condition_expression}")
   print(f"Operator Count: {condition_op_count}")

   print("\nExample 5: Testing the 300 operator limit")

   # This is just for demonstration - in a real application, you would use your
actual table
   # Note: This function is renamed to _test_expression_limit to avoid pytest
trying to run it
   print("In a real application, you would test with _test_expression_limit
function")
   print("Expression with 250 operators would be under the limit")
   print("Expression with 350 operators would exceed the 300 operator limit")

   print("\nOperator Counting Rules in DynamoDB:")
   print("1. Comparison Operators (=, <>, <, <=, >, >=): 1 operator each")
   print("2. Logical Operators (AND, OR, NOT): 1 operator each")
   print("3. BETWEEN: 2 operators (BETWEEN + AND)")
   print("4. IN: 1 operator (regardless of number of values)")
   print("5. Functions (attribute_exists, begins_with, etc.): 1 operator each")
   print("6. Arithmetic Operators (+, -): 1 operator each")
   print("7. SET assignments (=): 1 operator each")
   print("8. ADD and DELETE operations: 1 operator each")

   print("\nStrategies for Working Within the 300 Operator Limit:")
   print("1. Break operations into multiple requests")
   print("2. Use DynamoDB Transactions for complex operations")
   print("3. Optimize data model to reduce query complexity")
   print("4. Use application-side filtering for less critical filters")
   print("5. Consider using IN operator instead of multiple OR conditions")
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créer une API REST API Gateway pour suivre les données de la COVID-19

L'exemple de code suivant montre comment créer une API REST qui simule un système pour suivre les cas quotidiens de COVID-19 aux États-Unis, à l'aide de données fictives.

Python

Kit SDK for Python (Boto3)

Montre comment utiliser AWS Chalice avec le AWS SDK pour Python (Boto3) pour créer une API REST sans serveur qui utilise Amazon API Gateway et Amazon DynamoDB.AWS Lambda L'API REST simule un système qui suit les cas quotidiens de COVID-19 aux États-Unis à l'aide de données fictives. Découvrez comment :

- Utilisez AWS Chalice pour définir des routes dans les fonctions Lambda appelées pour gérer les requêtes REST qui passent par API Gateway.
- Utilisez les fonctions Lambda pour récupérer et stocker des données dans une table DynamoDB afin de répondre aux demandes REST.
- Définissez la structure des tables et les ressources des rôles de sécurité dans un AWS CloudFormation modèle.
- Utilisez AWS Chalice CloudFormation pour empaqueter et déployer toutes les ressources nécessaires.
- CloudFormation À utiliser pour nettoyer toutes les ressources créées.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- CloudFormation
- DynamoDB

- Lambda

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créer une application de messagerie avec Step Functions

L'exemple de code suivant montre comment créer une application de AWS Step Functions messagerie qui extrait les enregistrements de messages d'une table de base de données.

Python

Kit SDK for Python (Boto3)

Montre comment utiliser le AWS SDK pour Python (Boto3) with AWS Step Functions pour créer une application de messagerie qui récupère les enregistrements de messages d'une table Amazon DynamoDB et les envoie via Amazon Simple Queue Service (Amazon SQS). La machine d'état intègre une AWS Lambda fonction permettant de scanner la base de données à la recherche de messages non envoyés.

- Créez une machine d’état qui extrait et met à jour des enregistrements de message d’une table Amazon DynamoDB.
- Mettez à jour la définition de la machine d’état pour envoyer des messages à Amazon Simple Queue Service (Amazon SQS).
- Démarrez et arrêtez les exécutions de la machine.
- Connectez-vous à Lambda, DynamoDB et Amazon SQS à partir d’une machine d’état à l’aide d’intégrations de services.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Création d'une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l'aide d'étiquettes

Les exemples de code suivants montrent comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

.NET

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK pour C++

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK pour Java 2.x

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK pour Kotlin

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

Kit SDK pour PHP

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK pour Rust

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Création d'une table DynamoDB avec un index secondaire global à l'aide du SDK AWS

L'exemple de code suivant montre comment créer une table avec un index secondaire global.

Java

SDK pour Java 2.x

Créez une table DynamoDB avec un index secondaire global à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
```

```
import software.amazon.awssdk.services.dynamodb.model.ProjectionType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public void createTable() {
        try {
            // Attribute definitions
            final List<AttributeDefinition> attributeDefinitions = new
 ArrayList<>();
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(ISSUE_ID_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(TITLE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(CREATE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(DUE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());

            // Key schema for table
            final List<KeySchemaElement> tableKeySchema = new ArrayList<>();
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.HASH)
                .build()); // Partition key
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.RANGE)
```

```
                .build()); // Sort key

            // Initial provisioned throughput settings for the indexes
            final ProvisionedThroughput ptIndex = ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build();

            // CreateDateIndex
            final List<KeySchemaElement> createDateKeySchema = new ArrayList<>();
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(CREATE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection createDateProjection = Projection.builder()
                .projectionType(ProjectionType.INCLUDE)
                .nonKeyAttributes(DESCRIPTION_ATTR, STATUS_ATTR)
                .build();

            final GlobalSecondaryIndex createDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(CREATE_DATE_INDEX)
                .keySchema(createDateKeySchema)
                .projection(createDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // TitleIndex
            final List<KeySchemaElement> titleKeySchema = new ArrayList<>();
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection titleProjection =
```

```
Projection.builder().projectionType(ProjectionType.KEYS_ONLY).build();

            final GlobalSecondaryIndex titleIndex =
GlobalSecondaryIndex.builder()
                .indexName(TITLE_INDEX)
                .keySchema(titleKeySchema)
                .projection(titleProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // DueDateIndex
            final List<KeySchemaElement> dueDateKeySchema = new ArrayList<>();
            dueDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(DUE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());

            final Projection dueDateProjection =
                Projection.builder().projectionType(ProjectionType.ALL).build();

            final GlobalSecondaryIndex dueDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(DUE_DATE_INDEX)
                .keySchema(dueDateKeySchema)
                .projection(dueDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            final CreateTableRequest createTableRequest =
CreateTableRequest.builder()
                .tableName(TABLE_NAME)
                .keySchema(tableKeySchema)
                .attributeDefinitions(attributeDefinitions)
                .globalSecondaryIndexes(createDateIndex, titleIndex,
dueDateIndex)
                .provisionedThroughput(ProvisionedThroughput.builder()
                    .readCapacityUnits(1L)
                    .writeCapacityUnits(1L)
                    .build())
                .build();

            System.out.println("Creating table " + TABLE_NAME + "...");
            dynamoDbClient.createTable(createTableRequest);
```

```
            // Wait for table to become active
            System.out.println("Waiting for " + TABLE_NAME + " to become
ACTIVE...");
            final DynamoDbWaiter waiter = dynamoDbClient.waiter();
            final DescribeTableRequest describeTableRequest =
                DescribeTableRequest.builder().tableName(TABLE_NAME).build();

            final WaiterResponse<DescribeTableResponse> waiterResponse =
                waiter.waitUntilTableExists(describeTableRequest);
            waiterResponse.matched().response().ifPresent(response ->
System.out.println("Table is now ready for use"));

        } catch (DynamoDbException e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for Java 2.x API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Création d'une table DynamoDB avec un paramètre de débit à chaud à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment créer une table avec le débit à chaud activé.

Java

SDK pour Java 2.x

Création d’une table DynamoDB avec le paramètre de débit à chaud à l’aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
```

```
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

   public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
       return WarmThroughput.builder()
           .readUnitsPerSecond(readUnitsPerSecond)
           .writeUnitsPerSecond(writeUnitsPerSecond)
           .build();
   }

   /**
    * Builds a ProvisionedThroughput object with the specified read and write
 capacity units.
    *
    * @param readCapacityUnits The read capacity units
    * @param writeCapacityUnits The write capacity units
    * @return A configured ProvisionedThroughput object
    */
   public static ProvisionedThroughput buildProvisionedThroughput(
       final Long readCapacityUnits, final Long writeCapacityUnits) {
       return ProvisionedThroughput.builder()
           .readCapacityUnits(readCapacityUnits)
           .writeCapacityUnits(writeCapacityUnits)
           .build();
   }

   /**
    * Builds an AttributeDefinition with the specified name and type.
    *
    * @param attributeName The attribute name
    * @param scalarAttributeType The attribute type
    * @return A configured AttributeDefinition
    */
   private static AttributeDefinition buildAttributeDefinition(
       final String attributeName, final ScalarAttributeType
 scalarAttributeType) {
```

```
        return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
    }

    /**
     * Builds a KeySchemaElement with the specified name and key type.
     *
     * @param attributeName The attribute name
     * @param keyType The key type (HASH or RANGE)
     * @return A configured KeySchemaElement
     */
    private static KeySchemaElement buildKeySchemaElement(final String
attributeName, final KeyType keyType) {
        return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
    }

    /**
     * Creates a DynamoDB table with the specified configuration including warm
throughput settings.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to create
     * @param partitionKey The partition key attribute name
     * @param sortKey The sort key attribute name
     * @param miscellaneousKeyAttribute Additional key attribute name for GSI
     * @param nonKeyAttribute Non-key attribute to include in GSI projection
     * @param tableReadCapacityUnits Read capacity units for the table
     * @param tableWriteCapacityUnits Write capacity units for the table
     * @param tableWarmReadUnitsPerSecond Warm read units per second for the
table
     * @param tableWarmWriteUnitsPerSecond Warm write units per second for the
table
     * @param globalSecondaryIndexName The name of the GSI to create
     * @param globalSecondaryIndexReadCapacityUnits Read capacity units for the
GSI
     * @param globalSecondaryIndexWriteCapacityUnits Write capacity units for the
GSI
     * @param globalSecondaryIndexWarmReadUnitsPerSecond Warm read units per
second for the GSI
```

```
     * @param globalSecondaryIndexWarmWriteUnitsPerSecond Warm write units per
second for the GSI
     */
    public static void createDynamoDBTable(
        final DynamoDbClient ddb,
        final String tableName,
        final String partitionKey,
        final String sortKey,
        final String miscellaneousKeyAttribute,
        final String nonKeyAttribute,
        final Long tableReadCapacityUnits,
        final Long tableWriteCapacityUnits,
        final Long tableWarmReadUnitsPerSecond,
        final Long tableWarmWriteUnitsPerSecond,
        final String globalSecondaryIndexName,
        final Long globalSecondaryIndexReadCapacityUnits,
        final Long globalSecondaryIndexWriteCapacityUnits,
        final Long globalSecondaryIndexWarmReadUnitsPerSecond,
        final Long globalSecondaryIndexWarmWriteUnitsPerSecond) {

        // Define the table attributes
        final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
        final AttributeDefinition sortKeyAttribute =
buildAttributeDefinition(sortKey, ScalarAttributeType.S);
        final AttributeDefinition miscellaneousKeyAttributeDefinition =
            buildAttributeDefinition(miscellaneousKeyAttribute,
ScalarAttributeType.N);
        final AttributeDefinition[] attributeDefinitions = {
            partitionKeyAttribute, sortKeyAttribute,
miscellaneousKeyAttributeDefinition
        };

        // Define the table key schema
        final KeySchemaElement partitionKeyElement =
buildKeySchemaElement(partitionKey, KeyType.HASH);
        final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
        final KeySchemaElement[] keySchema = {partitionKeyElement,
sortKeyElement};

        // Define the provisioned throughput for the table
        final ProvisionedThroughput provisionedThroughput =
```

```
            buildProvisionedThroughput(tableReadCapacityUnits,
tableWriteCapacityUnits);

        // Define the Global Secondary Index (GSI)
        final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
        final KeySchemaElement globalSecondaryIndexSortKeyElement =
            buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
        final KeySchemaElement[] gsiKeySchema = {
            globalSecondaryIndexPartitionKeyElement,
globalSecondaryIndexSortKeyElement
        };

        final Projection gsiProjection = Projection.builder()
            .projectionType(PROJECTION_TYPE_INCLUDE)
            .nonKeyAttributes(nonKeyAttribute)
            .build();

        final ProvisionedThroughput gsiProvisionedThroughput =
            buildProvisionedThroughput(globalSecondaryIndexReadCapacityUnits,
globalSecondaryIndexWriteCapacityUnits);

        // Define the warm throughput for the Global Secondary Index (GSI)
        final WarmThroughput gsiWarmThroughput = buildWarmThroughput(
            globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);

        final GlobalSecondaryIndex globalSecondaryIndex =
GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

        // Define the warm throughput for the table
        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableWarmReadUnitsPerSecond,
tableWarmWriteUnitsPerSecond);

        final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
```

```
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

        final CreateTableResponse response = ddb.createTable(request);
        System.out.println(response);
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Créez une table DynamoDB avec le paramètre de débit à chaud à l’aide du kit AWS SDK pour JavaScript.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
```

```
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
```

```
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Créez une table DynamoDB avec le paramètre de débit à chaud à l’aide du kit AWS SDK pour Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def create_dynamodb_table_warm_throughput(
    table_name,
    partition_key,
    sort_key,
    misc_key_attr,
    non_key_attr,
```

```
    table_provisioned_read_units,
    table_provisioned_write_units,
    table_warm_reads,
    table_warm_writes,
    gsi_name,
    gsi_provisioned_read_units,
    gsi_provisioned_write_units,
    gsi_warm_reads,
    gsi_warm_writes,
    region_name="us-east-1",
):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being
 created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned
 read capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's
 warm throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
    :param gsi_name: The name of the Global Secondary Index (GSI) to be created
 on the table.
    :param gsi_provisioned_read_units: The configured Global Secondary Index
 (GSI) provisioned read capacity units.
    :param gsi_provisioned_write_units: The configured Global Secondary Index
 (GSI) provisioned write capacity units.
    :param gsi_warm_reads: The read units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param gsi_warm_writes: The write units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = client("dynamodb", region_name=region_name)

        # Define the table attributes
```

```
        attribute_definitions = [
            {"AttributeName": partition_key, "AttributeType": "S"},
            {"AttributeName": sort_key, "AttributeType": "S"},
            {"AttributeName": misc_key_attr, "AttributeType": "N"},
        ]

        # Define the table key schema
        key_schema = [
            {"AttributeName": partition_key, "KeyType": "HASH"},
            {"AttributeName": sort_key, "KeyType": "RANGE"},
        ]

        # Define the provisioned throughput for the table
        provisioned_throughput = {
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units,
        }

        # Define the global secondary index
        gsi_key_schema = [
            {"AttributeName": sort_key, "KeyType": "HASH"},
            {"AttributeName": misc_key_attr, "KeyType": "RANGE"},
        ]
        gsi_projection = {"ProjectionType": "INCLUDE", "NonKeyAttributes":
[non_key_attr]}
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units,
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes,
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput,
            }
        ]

        # Define the warm throughput for the table
```

```
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes,
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput,
        )

        print(response)
        return response
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

- Pour plus de détails sur l'API, consultez CreateTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créer une application web pour suivre les données DynamoDB

Les exemples de code suivants montrent comment créer une application web qui suit des éléments de travail dans une table Amazon DynamoDB et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

.NET

SDK pour .NET

Montre comment utiliser l'API Amazon DynamoDB .NET pour créer une application web dynamique qui suit les données de travail DynamoDB.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Java

SDK pour Java 2.x

Montre comment utiliser l'API Amazon DynamoDB pour créer une application web dynamique qui suit les données de travail DynamoDB.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Kotlin

SDK pour Kotlin

Montre comment utiliser l'API Amazon DynamoDB pour créer une application web dynamique qui suit les données de travail DynamoDB.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Python

SDK pour Python (Boto3)

Montre comment utiliser le AWS SDK pour Python (Boto3) pour créer un service REST qui suit les éléments de travail dans Amazon DynamoDB et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise la structure web Flask pour gérer le routage HTTP et s'intègre à une page web React pour présenter une application web entièrement fonctionnelle.

- Créez un service Flask REST qui s'intègre àServices AWS.
- Lisez, écrivez et mettez à jour les éléments de travail stockés dans une table DynamoDB.
- Utilisez Amazon SES pour envoyer des rapports par e-mail sur les éléments de travail.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet dans le référentiel d'exemples de AWS code sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créer une application de chat Websocket avec API Gateway

L'exemple de code suivant montre comment créer une application de chat desservie par une API Websocket basée sur Amazon API Gateway.

Python

Kit SDK for Python (Boto3)

Montre comment utiliser Amazon API Gateway V2 pour créer une API Websocket qui s'intègre à Amazon AWS Lambda DynamoDB.AWS SDK pour Python (Boto3)

- Créez une API WebSocket dans API Gateway.
- Définissez un gestionnaire Lambda qui stocke les connexions dans DynamoDB et publie des messages pour d'autres participants au chat.

- Connectez-vous à l’application de chat Websocket et envoyez des messages avec le package Websockets.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Création d'un élément DynamoDB avec un TTL à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment créer un élément avec un TTL.

Java

SDK pour Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
```

```
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
```

```
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
```

```
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
```

```
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };
```

```
    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.
```

```
    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Pour plus de détails sur l'API, consultez PutItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Créez et gérez des tables globales DynamoDB avec une forte cohérence multirégionale à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment créer et gérer des tables globales DynamoDB avec MRSC (forte cohérence multirégionale).

- Création d'une table avec une forte cohérence multirégionale.
- Vérifiez la configuration MRSC et le statut du réplica.
- Testez une forte cohérence entre les régions grâce à des lectures immédiates.
- Effectuez des écritures conditionnelles avec les garanties MRSC.
- Nettoyez les ressources des tables globales MRSC.

Bash

AWS CLI avec le script Bash

Créez une table avec une forte cohérence multirégionale.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
```

```
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Vérifiez la configuration MRSC et le statut du réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Testez une forte cohérence avec des lectures immédiates dans les régions.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Effectuez des écritures conditionnelles avec les garanties MRSC.

```
# Perform a conditional update from a different region
```

```
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Nettoyez les ressources des tables globales MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK pour Java 2.x

Créez un tableau régional prêt pour la conversion MRSC à l'aide AWS SDK for Java 2.x de.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
                       .keyType(KeyType.RANGE)
                       .build())
               .billingMode(BillingMode.PAY_PER_REQUEST)
               .build();
```

```
            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Convertissez une table régionale en MRSC à l'aide de répliques et de témoins à l'aide de témoins.AWS SDK for Java 2.x

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
```

```
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Décrivez une configuration de table globale MRSC à l'aide AWS SDK for Java 2.x de.

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           if (response.table().globalTableWitnesses() != null
               && !response.table().globalTableWitnesses().isEmpty()) {
               LOGGER.info("Number of witnesses: "
                   + response.table().globalTableWitnesses().size());
```

```
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Ajout d'éléments de test pour vérifier la forte cohérence du MRSC à l'aide du kit AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lisez des éléments avec des lectures cohérentes à partir de répliques MRSC à l'aide de.AWS SDK for Java 2.x

```
   public static GetItemResponse getItemWithConsistentRead(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

           Map<String, AttributeValue> key = new HashMap<>();
           key.put("Artist", AttributeValue.builder().s(artist).build());
           key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           GetItemRequest getItemRequest = GetItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .consistentRead(true)
               .build();

           GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

           if (response.hasItem()) {
               LOGGER.info("Item found with strong consistency - no wait time
needed");
           } else {
               LOGGER.info("Item not found");
```

```
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Effectuez des mises à jour conditionnelles avec les garanties MRSC en utilisant AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }
```

```
        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Attendez que les répliques du MRSC et que les témoins deviennent actifs en utilisant.AWS SDK for Java 2.x

```
   public static void waitForMRSCReplicasActive(
       final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
       throws InterruptedException {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (maxWaitTimeSeconds <= 0) {
           throw new IllegalArgumentException("Max wait time must be positive");
       }

       try {
           LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

           final long startTime = System.currentTimeMillis();
           final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
           int backoffSeconds = 5; // Start with 5 second intervals
           final int maxBackoffSeconds = 30; // Cap at 30 seconds

           while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
               DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

               boolean allActive = true;
               StringBuilder statusReport = new StringBuilder();

               if (response.table().multiRegionConsistency() == null
                   || !MultiRegionConsistency.STRONG
                       .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                   allActive = false;
```

```
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
```

```
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Nettoyez les répliques du MRSC et les témoins à l'aide de.AWS SDK for Java 2.x

```
   public static UpdateTableResponse cleanupMRSCReplicas(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

           // Remove replica using ReplicationGroupUpdate
           ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
               .delete(DeleteReplicationGroupMemberAction.builder()
                   .regionName(replicaRegion.id())
                   .build())
               .build();

           // Remove witness
           GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
               .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                   .regionName(witnessRegion.id())
                   .build())
               .build();

           UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
               .tableName(tableName)
```

```
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Démonstration complète du flux de travail MRSC à l'aide de AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
```

```
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
```

```
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
```

```
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK for Java 2.x.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Création et gestion de tables globales DynamoDB illustrant MREC à l'aide d'un SDK AWS

L'exemple de code suivant montre comment créer et gérer des tables globales DynamoDB avec des réplicas entre plusieurs régions.

- Création d'une table avec un index secondaire global et DynamoDB Streams.
- Ajout de réplicas dans différentes régions pour créer une table globale.
- Suppression de réplicas d'une table globale.
- Ajout d'éléments de test pour vérifier la réplication entre plusieurs régions.
- Description de la configuration globale de la table et de l'état du réplica.

Java

SDK pour Java 2.x

Créez une table avec l'index secondaire global et les flux DynamoDB à l'aide de.AWS SDK for Java 2.x

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
```

```
            LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Attendez qu'une table soit active en utilisant AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Ajoutez une réplique pour créer ou étendre une table globale à l'aide de AWS SDK for Java 2.x.

```
   public static UpdateTableResponse addReplica(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final String indexName,
       final Long readCapacity) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }
       if (readCapacity == null || readCapacity <= 0) {
           throw new IllegalArgumentException("Read capacity must be a positive
number");
       }

       try {
           LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

           // Create a ReplicationGroupUpdate for adding a replica
           ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
               .create(builder -> builder.regionName(replicaRegion.id())
                   .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                       .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                           .readCapacityUnits(readCapacity)
                           .build())
```

```
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Supprimez une réplique d'une table globale à l'aide de AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
```

```
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Ajoutez des éléments de test pour vérifier la réplication à l'aide de AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
```

```
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Décrivez la configuration globale des tables et les répliques à l'aide AWS SDK for Java 2.x de.

```
   public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           return response;

       } catch (ResourceNotFoundException e) {
           LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
           throw e;
```

```
        }
    }
```

Exemple complet d'opérations de table globales utilisant AWS SDK for Java 2.x.

```
   public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");
```

```
                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK for Java 2.x.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Supprimer des données DynamoDB à l'aide des instructions PARTIQL DELETE avec un SDK AWS

L'exemple de code suivant montre comment supprimer des données à l'aide des instructions PartiQL DELETE.

JavaScript

SDK pour JavaScript (v3)

Supprimez des éléments d'une table DynamoDB à l'aide des instructions PARTIQL DELETE avec.AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
```

```
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};
```

```
/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
 AND ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
```

```
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Détectez le PPE dans les images avec Amazon Rekognition à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment créer une application qui utilise Amazon Rekognition afin de détecter l'équipement de protection individuelle (EPI) dans les images.

Java

SDK pour Java 2.x

Montre comment créer une AWS Lambda fonction qui détecte les images à l'aide d'un équipement de protection individuelle.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Insérer des données DynamoDB à l'aide des instructions PartiQL INSERT avec un SDK AWS

L'exemple de code suivant montre comment insérer des données à l'aide des instructions PartiQL INSERT.

JavaScript

SDK pour JavaScript (v3)

Insérez des éléments dans une table DynamoDB à l'aide des instructions PartiQL INSERT avec.AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) =>
 {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
```

```
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
```

```
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
```

```
// Example table with a simple primary key (just partition key)
const simpleKeyItem = {
  userId: "user123",
  name: "John Doe",
  email: "john@example.com",
};
await insertItem("UsersTable", simpleKeyItem);

// Example table with composite key (partition key + sort key)
const compositeKeyItem = {
  orderId: "order456",
  productId: "prod789",
  quantity: 2,
  price: 29.99,
};
await insertItem("OrdersTable", compositeKeyItem);

// Example with Global Secondary Index (GSI)
// The GSI might be on the email attribute
const gsiItem = {
  userId: "user789",
  email: "jane@example.com",
  name: "Jane Smith",
  userType: "premium",  // This could be part of a GSI
};
await insertItem("UsersTable", gsiItem);

// Example with Local Secondary Index (LSI)
// LSI uses the same partition key but different sort key
const lsiItem = {
  orderId: "order567",  // Partition key
  productId: "prod123", // Sort key for the table
  orderDate: "2023-11-15", // Potential sort key for an LSI
  quantity: 1,
  price: 19.99,
};
await insertItem("OrdersTable", lsiItem);

// Batch insert example with multiple items
const batchItems = [
  {
    userId: "user234",
    name: "Alice Johnson",
    email: "alice@example.com",
```

```
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Invoquer une fonction Lambda à partir d'un navigateur

L'exemple de code suivant montre comment invoquer une AWS Lambda fonction depuis un navigateur.

JavaScript

SDK pour JavaScript (v2)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda

SDK pour JavaScript (v3)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs. Cette application utilise la AWS SDK pour JavaScript version 3.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Gestion des index secondaires globaux DynamoDB à l'aide de la version 2 AWS Command Line Interface

L'exemple de code suivant montre comment gérer le cycle de vie complet des index secondaires globaux.

- Créez une table avec un index secondaire global.
- Ajoutez un nouveau GSI à une table existante.
- Mettez à jour (augmentez) le débit à chaud du GSI.
- Interrogez les données en utilisant GSIs.
- Supprimez un GSI.

Bash

AWS CLI avec le script Bash

Créez une table avec un index secondaire global.

```
# Create a table with a GSI
aws dynamodb create-table \
```

```
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "IndexName=AlbumIndex,\
        KeySchema=[{AttributeName=AlbumTitle,KeyType=HASH}],\
        Projection={ProjectionType=ALL}"
```

Ajoutez un nouveau GSI (à la demande) à une table existante.

```
# Add a new GSI to an existing table
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Genre,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\":\"GenreIndex\",\
        \"KeySchema\":[{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"}],\
        \"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

Mettez à jour (augmentez) le débit à chaud du GSI.

```
# Increase the warm throughput of a GSI (default values are 12k reads, 4k writes)
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Update\":{\"IndexName\":\"AlbumIndex\",\
        \"WarmThroughput\":{\"ReadUnitsPerSecond\":15000,\"WriteUnitsPerSecond
\":6000}}}]"
```

Interrogez les données en utilisant GSIs.

```
# Query the AlbumIndex GSI
```

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Let It Be"}}'

# Query the GenreIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name GenreIndex \
    --key-condition-expression "Genre = :genre" \
    --expression-attribute-values '{":genre":{"S":"Jazz"}}'
```

Supprimez un GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - DeleteTable
  - Interrogation
  - UpdateTable

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Gérez les politiques basées sur les ressources DynamoDB à l'aide de la version 2 AWS Command Line Interface

L'exemple de code suivant montre comment gérer le cycle de vie complet des politiques basées sur les ressources pour les tables DynamoDB.

- Créez une table avec une politique de ressources.
- Obtenez une politique de ressources.
- Mettez à jour une politique de ressources.
- Supprimez une politique de ressources.

Bash

AWS CLI avec le script Bash

Créez une table avec une politique de ressources.

```
# Step 1: Create a DynamoDB table
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST

# Step 2: Create a resource-based policy document
cat > policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/DynamoDBReadOnly"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
```

```
EOF

# Step 3: Attach the resource-based policy to the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://policy.json
```

Obtenez une politique de ressources.

```
# Get the resource-based policy attached to a table
aws dynamodb get-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Mettez à jour une politique de ressources.

```
# Step 1: Create an updated policy document
cat > updated-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:role/DynamoDBReadOnly",
          "arn:aws:iam::123456789012:role/DynamoDBAnalytics"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF
```

```
# Step 2: Update the resource-based policy on the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://updated-policy.json
```

Supprimez une politique de ressources.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Surveillez les performances d'Amazon DynamoDB à l'aide d'un SDK AWS

L'exemple de code suivant montre comment configurer l'utilisation de DynamoDB par une application afin d'en surveiller les performances.

Java

SDK pour Java 2.x

Cet exemple montre comment configurer une application Java pour surveiller les performances de DynamoDB. L'application envoie des données métriques vers CloudWatch lesquelles vous pouvez surveiller les performances.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- CloudWatch
- DynamoDB

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Exécuter des opérations de requête DynamoDB avancées à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment exécuter des opérations de requête avancées dans DynamoDB.

- Interrogation des tables à l’aide de diverses techniques de filtrage et de conditions.
- Mettez en œuvre la pagination pour les grands ensembles de résultats.
- Utilisation d’index secondaires globaux pour les autres modèles d’accès.
- Appliquez des contrôles de cohérence en fonction des exigences de l’application.

Java

SDK pour Java 2.x

Requête avec des lectures très cohérentes utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
   public QueryResponse queryWithConsistentReads(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final boolean useConsistentRead) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .consistentRead(useConsistentRead)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
           return response;
       } catch (ResourceNotFoundException e) {
           LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
           throw e;
       }
   }
```

Requête à l'aide d'un index secondaire global avec AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
```

```
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
```

```
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Requête avec pagination à l'aide AWS SDK for Java 2.x de.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }
```

```
                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

Requête à l'aide de filtres complexes à l'aide de AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
 Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
```

```
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

Requête avec une expression de filtre construite dynamiquement à l'aide de AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
 e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
 e.getMessage());
            throw e;
```

```
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
```

```
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

Requête avec une expression de filtre et limitation de l'utilisation AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Requête avec des lectures très cohérentes utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Requête à l'aide d'un index secondaire global avec AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
```

```
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Requête avec pagination à l'aide AWS SDK pour JavaScript de.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
```

```
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}
```

```
/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

Requête à l'aide de filtres complexes à l'aide de AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

Requête avec une expression de filtre construite dynamiquement à l'aide de AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
```

```
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Requête avec des lectures très cohérentes utilisant AWS SDK pour Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
```

```
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

Requête à l'aide d'un index secondaire global avec AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

Requête avec pagination à l'aide AWS SDK pour Python (Boto3) de.

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
```

```
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
```

```
    """
    Query a DynamoDB table with pagination using a generator to handle large
result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
```

```
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

Requête à l'aide de filtres complexes à l'aide de AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition
```

```
    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)
```

```
    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
```

```
    response = table.query(**query_params)
    return response
```

Requête avec une expression de filtre construite dynamiquement à l'aide de AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}
```

```
    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Requête avec une expression de filtre et limitation de l'utilisation AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}
```

```
    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Effectuez des opérations de liste dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment exécuter des opérations de liste dans DynamoDB.

- Ajout d'éléments à un attribut de liste.
- Supprimez des éléments d'un attribut de liste.
- Mettez à jour des éléments spécifiques dans une liste par index.
- Utilisez des fonctions d'ajout de liste et d'index de liste.

Java

SDK pour Java 2.x

Démontrez les opérations de liste en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Appends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the end of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToAppend The items to append to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse appendToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToAppend) {

        // Create a list value from the items to append
        AttributeValue listValue =
 AttributeValue.builder().l(itemsToAppend).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName =
 list_append(if_not_exists(#attrName, :emptyList), :newItems)")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
```

```
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Prepends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the beginning of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToPrepend The items to prepend to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse prependToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToPrepend) {

        // Create a list value from the items to prepend
        AttributeValue listValue =
AttributeValue.builder().l(itemsToPrepend).build();

        // Define the update parameters
        // Note: To prepend, we put the new items first in the list_append
function
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
```

```
            .updateExpression("SET #attrName = list_append(:newItems,
if_not_exists(#attrName, :emptyList))")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates a specific element in a list attribute.
     *
     * <p>This method demonstrates how to update a specific element in a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to update
     * @param newValue The new value for the element
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index,
        AttributeValue newValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName[" + index + "] = :newValue")
```

```
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(":newValue", newValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes a specific element from a list attribute.
     *
     * <p>This method demonstrates how to remove a specific element from a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #attrName[" + index + "]")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
```

```
     * Gets the current value of a list attribute.
     *
     * <p>Helper method to retrieve the current value of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param listAttributeName The name of the list attribute
     * @return The list attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static List<AttributeValue> getListAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String listAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(listAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the list attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(listAttributeName)) {
                return response.item().get(listAttributeName).l();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get list attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemple d'utilisation d'opérations de liste avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating list operations in DynamoDB");

       try {
           // Example 1: Append items to a list
           System.out.println("\nExample 1: Appending items to a list");
           List<AttributeValue> tagsToAppend = List.of(
               AttributeValue.builder().s("Electronics").build(),
               AttributeValue.builder().s("Gadget").build());

           UpdateItemResponse appendResponse = appendToList(dynamoDbClient,
tableName, key, "Tags", tagsToAppend);

           System.out.println("Updated list attribute: " +
appendResponse.attributes());

           // Example 2: Prepend items to a list
           System.out.println("\nExample 2: Prepending items to a list");
           List<AttributeValue> tagsToPrepend = List.of(
               AttributeValue.builder().s("Featured").build(),
               AttributeValue.builder().s("New").build());

           UpdateItemResponse prependResponse = prependToList(dynamoDbClient,
tableName, key, "Tags", tagsToPrepend);

           System.out.println("Updated list attribute: " +
prependResponse.attributes());

           // Example 3: Update a specific element in a list
           System.out.println("\nExample 3: Updating a specific element in a
list");
           UpdateItemResponse updateResponse = updateListElement(
               dynamoDbClient,
               tableName,
               key,
               "Tags",
               0,
               AttributeValue.builder().s("BestSeller").build());
```

```
            System.out.println("Updated list attribute: " +
updateResponse.attributes());

            // Example 4: Remove a specific element from a list
            System.out.println("\nExample 4: Removing a specific element from a
list");
            UpdateItemResponse removeResponse = removeListElement(dynamoDbClient,
tableName, key, "Tags", 1);

            System.out.println("Updated list attribute: " +
removeResponse.attributes());

            // Example 5: Get the current value of a list attribute
            System.out.println("\nExample 5: Getting the current value of a list
attribute");
            List<AttributeValue> currentList = getListAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentList != null) {
                System.out.println("Current list attribute:");
                for (int i = 0; i < currentList.size(); i++) {
                    System.out.println("  [" + i + "]: " +
currentList.get(i).s());
                }
            } else {
                System.out.println("List attribute not found");
            }

            // Explain list operations
            System.out.println("\nKey points about DynamoDB list operations:");
            System.out.println("1. Lists are ordered collections of attributes");
            System.out.println("2. Use list_append to add items to a list");
            System.out.println("3. To append items, use list_append(existingList,
newItems)");
            System.out.println("4. To prepend items, use list_append(newItems,
existingList)");
            System.out.println("5. Use index notation (list[0]) to access or
update specific elements");
            System.out.println("6. Use REMOVE to delete elements from a list");
            System.out.println("7. List indices are zero-based");
            System.out.println("8. Use if_not_exists to handle the case where the
list doesn't exist yet");
```

```
        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez les opérations de liste en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
```

```
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list
 structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));
```

```
  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations de liste avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);
```

```
    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
    console.log("\nCurrent state of RecentSearches:",
currentItem?.RecentSearches);

    // Example 3: Update a specific element in a list
    console.log("\nExample 3: Updating a specific element in a list");
    const response3 = await updateListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      0, // Update the first element
      "mechanical keyboard" // New value
    );

    console.log("Updated list element:", response3.Attributes);

    // Example 4: Remove an element from a list
    console.log("\nExample 4: Removing an element from a list");
    const response4 = await removeListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      2 // Remove the third element
    );

    console.log("List after removing element:", response4.Attributes);

    // Example 5: Create and concatenate lists
    console.log("\nExample 5: Creating and concatenating lists");

    // First, create two separate lists
    await updateWithMultipleActions(
      config,
      tableName,
      key,
      "SET WishList = :wishlist, SavedItems = :saveditems",
      null,
      {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
```

```
    }
  );

  // Then, concatenate them
  const response5 = await concatenateLists(
    config,
    tableName,
    key,
    "WishList",
    "SavedItems",
    "AllItems"
  );

  console.log("Concatenated lists:", response5.Attributes);

  // Example 6: Create a nested list structure
  console.log("\nExample 6: Creating a nested list structure");
  const response6 = await createNestedList(
    config,
    tableName,
    key,
    "Categories",
    [
      ["Electronics", "Computers", "Accessories"],
      ["Books", "Magazines", "E-books"],
      ["Clothing", "Shoes", "Watches"]
    ]
  );

  console.log("Created nested list:", response6.Attributes);

  // Example 7: Update an element in a nested list
  console.log("\nExample 7: Updating an element in a nested list");
  const response7 = await updateNestedListElement(
    config,
    tableName,
    key,
    "Categories",
    0, // First inner list
    1, // Second element in that list
    "Laptops" // New value
  );

  console.log("Updated nested list element:", response7.Attributes);
```

```
    // Get the final state of the item
    currentItem = await getItem(config, tableName, key);
    console.log("\nFinal state of the item:", JSON.stringify(currentItem, null,
 2));

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
 newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not
 exist yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a
 list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested
 list elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name
 placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
```

```
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez les opérations de liste en utilisant AWS SDK pour Python (Boto3).

```
import boto3
import json
from typing import Any, Dict, List, Optional, Union


def create_list_attribute(
    table_name: str, key: Dict[str, Any], list_name: str, list_values: List[Any]
) -> Dict[str, Any]:
    """
    Create a new list attribute or replace an existing one.

    This function demonstrates how to create a new list attribute or replace
    an existing list with new values.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        list_values (List[Any]): The values to set in the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation to create or replace the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = :list_values",
        ExpressionAttributeValues={":list_values": list_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_append:
 List[Any]
) -> Dict[str, Any]:
```

```
    """
    Append values to the end of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the end of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append to add values to the end of the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append({list_name}, :values)",
        ExpressionAttributeValues={":values": values_to_append},
        ReturnValues="UPDATED_NEW",
    )

    return response


def prepend_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_prepend:
 List[Any]
) -> Dict[str, Any]:
    """
    Prepend values to the beginning of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the beginning of a list attribute.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_prepend (List[Any]): The values to prepend to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append with reversed order to add values to the beginning of the
 list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append(:values, {list_name})",
        ExpressionAttributeValues={":values": values_to_prepend},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int, new_value:
 Any
) -> Dict[str, Any]:
    """
    Update a specific element in a list attribute.

    This function demonstrates how to update a specific element in a list
 attribute
    using the index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to update.
        new_value (Any): The new value for the element.

    Returns:
```

```
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the index notation to update a specific element
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name}[{index}] = :value",
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int
) -> Dict[str, Any]:
    """
    Remove a specific element from a list attribute.

    This function demonstrates how to remove a specific element from a list
 attribute
    using the REMOVE action with index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to remove.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the REMOVE action with index notation to remove a specific element
    response = table.update_item(
```

```
        Key=key, UpdateExpression=f"REMOVE {list_name}[{index}]",
 ReturnValues="UPDATED_NEW"
    )

    return response


def update_nested_list_element(
    table_name: str, key: Dict[str, Any], path: str, new_value: Any
) -> Dict[str, Any]:
    """
    Update an element in a nested list structure.

    This function demonstrates how to update an element in a nested list
 structure
    using expression attribute names for the path components.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested element (e.g., "parent[0].child[1]").
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Define a type for path parts
    path_part = Dict[str, Union[str, int]]
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Parse the path to extract attribute names and indices
    path_parts: List[path_part] = []
    current_part = ""
    in_bracket = False

    for char in path:
        if char == "[":
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
            in_bracket = True
```

```
        elif char == "]":
            if current_part:
                # Fix for mypy: Use a properly typed dictionary with Union type
                path_parts.append({"type": "index", "value": int(current_part)})
                current_part = ""
            in_bracket = False
        elif char == "." and not in_bracket:
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
        else:
            current_part += char

    if current_part:
        path_parts.append({"type": "attribute", "value": current_part})

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        if part["type"] == "attribute":
            name_placeholder = f"#attr{i}"
            expression_attribute_names[name_placeholder] = part["value"]

            if path_expression:
                path_expression += "."
            path_expression += name_placeholder
        elif part["type"] == "index":
            path_expression += f"[{part['value']}]"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def create_list_if_not_exists(
    table_name: str, key: Dict[str, Any], list_name: str, default_values:
 List[Any]
) -> Dict[str, Any]:
    """
    Create a list attribute if it doesn't exist.

    This function demonstrates how to use if_not_exists to create a list
 attribute
    with default values if it doesn't already exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        default_values (List[Any]): The default values for the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to create the list if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 if_not_exists({list_name}, :default)",
        ExpressionAttributeValues={":default": default_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list_safely(
    table_name: str,
    key: Dict[str, Any],
```

```
    list_name: str,
    values_to_append: List[Any],
    default_values: Optional[List[Any]] = None,
) -> Dict[str, Any]:
    """
    Append values to a list, creating it if it doesn't exist.

    This function demonstrates how to safely append values to a list attribute,
    creating the list with default values if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
        default_values (Optional[List[Any]]): The default values if the list
 doesn't exist.
            If not provided, values_to_append will be used as the default.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # If default_values is not provided, use values_to_append
    if default_values is None:
        default_values = values_to_append

    # Use if_not_exists with list_append to safely append to the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 list_append(if_not_exists({list_name}, :default), :values)",
        ExpressionAttributeValues={
            ":default": default_values if default_values else [],
            ":values": values_to_append,
        },
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Exemple d'utilisation d'opérations de liste avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use list operations in DynamoDB."""
    # Example parameters
    table_name = "UserData"
    key = {"UserId": "user123"}

    print("Example 1: Creating a list attribute")
    try:
        response = create_list_attribute(
            table_name=table_name,
            key=key,
            list_name="Interests",
            list_values=["Reading", "Hiking", "Photography"],
        )
        print(
            f"List attribute created successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list attribute: {e}")

    print("\nExample 2: Appending values to a list")
    try:
        response = append_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_append=["Cooking", "Gardening"],
        )
        print(
            f"Values appended to list successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error appending to list: {e}")

    print("\nExample 3: Prepending values to a list")
```

```
    try:
        response = prepend_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_prepend=["Travel", "Music"],
        )
        print(
            f"Values prepended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error prepending to list: {e}")

    print("\nExample 4: Updating a specific list element")
    try:
        response = update_list_element(
            table_name=table_name,
            key=key,
            list_name="Interests",
            index=2,
            new_value="Mountain Hiking",
        )
        print(
            f"List element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating list element: {e}")

    print("\nExample 5: Removing a list element")
    try:
        response = remove_list_element(
            table_name=table_name, key=key, list_name="Interests", index=0
        )
        print(
            f"List element removed successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error removing list element: {e}")

    print("\nExample 6: Working with nested lists")
    try:
```

```
        # First, create an item with a nested structure
        dynamodb = boto3.resource("dynamodb")
        table = dynamodb.Table(table_name)

        table.update_item(
            Key={"UserId": "user456"},
            UpdateExpression="SET #skills = :skills",
            ExpressionAttributeNames={"#skills": "Skills"},
            ExpressionAttributeValues={
                ":skills": [
                    {"Category": "Programming", "Languages": ["Python", "Java",
"JavaScript"]},
                    {"Category": "Database", "Systems": ["DynamoDB", "MongoDB",
"PostgreSQL"]},
                ]
            },
        )

        # Now update a nested element
        response = update_nested_list_element(
            table_name=table_name,
            key={"UserId": "user456"},
            path="Skills[0].Languages[1]",
            new_value="TypeScript",
        )
        print(
            f"Nested list element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error working with nested lists: {e}")

    print("\nExample 7: Creating a list if it doesn't exist")
    try:
        response = create_list_if_not_exists(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Preferences",
            default_values=["Default1", "Default2", "Default3"],
        )
        print(
            f"List created with default values:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
```

```
    except Exception as e:
        print(f"Error creating list with default values: {e}")

    print("\nExample 8: Safely appending to a list")
    try:
        response = append_to_list_safely(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Notifications",
            values_to_append=["New message received"],
            default_values=[],
        )
        print(f"Safely appended to list: {json.dumps(response.get('Attributes',
{}), default=str)}")
    except Exception as e:
        print(f"Error safely appending to list: {e}")

    print("\nKey Points About Working with Lists in DynamoDB:")
    print("1. Lists are ordered collections of elements that can be of different
types")
    print("2. Use the SET operation with direct assignment to create or replace a
list")
    print("3. Use list_append() to add elements to a list without replacing the
entire list")
    print("4. To append to the end: list_append(list_name, :values)")
    print("5. To prepend to the beginning: list_append(:values, list_name)")
    print("6. Use index notation list_name[index] to access or update specific
elements")
    print("7. Use the REMOVE action with index notation to remove specific
elements")
    print("8. Lists can contain nested structures like maps and other lists")
    print("9. Use if_not_exists() to create a list with default values if it
doesn't exist")
    print("10. List indices are zero-based (the first element is at index 0)")
    print("11. Attempting to access an index beyond the list bounds will result
in an error")
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Effectuez des opérations cartographiques dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment exécuter des opérations de mappage dans DynamoDB.

- Ajout et mise à jour des attributs imbriqués dans les structures de mappage.
- Suppression des champs spécifiques des mappages.
- Utilisation d'attributs de mappage profondément imbriqués.

Java

SDK pour Java 2.x

Démontrez les opérations cartographiques en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates a map attribute that may not exist.
     *
     * <p>This method demonstrates how to safely update a map attribute
     * by using if_not_exists to handle the case where the map doesn't exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
```

```
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateMapAttributeSafe(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey,
        AttributeValue value) {

        // Create an empty map to use if the map doesn't exist
        Map<String, AttributeValue> emptyMap = new HashMap<>();
        AttributeValue emptyMapValue =
AttributeValue.builder().m(emptyMap).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = if_not_exists(#mapName, :emptyMap),
#mapName.#mapKey = :value")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .expressionAttributeValues(Map.of(
                ":value",
                value,
                ":emptyMap",
                AttributeValue.builder().m(new HashMap<>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds an attribute to a nested map.
     *
     * <p>This method demonstrates how to update a nested attribute without
```

```
     * overwriting the entire map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param path The path to the nested attribute as a list
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNestedMap(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> path,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < path.size(); i++) {
            expressionAttributeNames.put("#attr" + i, path.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
```

```
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes an attribute from a map.
     *
     * <p>This method demonstrates how to remove a specific attribute from a map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeMapAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #mapName.#mapKey")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Creates a map with multiple attributes in a single operation.
     *
     * <p>This method demonstrates how to create a map with multiple attributes
     * in a single update operation.
     *
```

```
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param attributes The attributes to set in the map
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createMapWithAttributes(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        Map<String, AttributeValue> attributes) {

        // Create a map value from the attributes
        AttributeValue mapValue = AttributeValue.builder().m(attributes).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = :mapValue")
            .expressionAttributeNames(Map.of("#mapName", mapName))
            .expressionAttributeValues(Map.of(":mapValue", mapValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a map attribute.
     *
     * <p>Helper method to retrieve the current value of a map attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param mapName The name of the map attribute
     * @return The map attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
```

```
   public static Map<String, AttributeValue> getMapAttribute(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String mapName) {

       // Define the get parameters
       GetItemRequest request = GetItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .projectionExpression(mapName)
           .build();

       try {
           // Perform the get operation
           GetItemResponse response = dynamoDbClient.getItem(request);

           // Return the map attribute if it exists, otherwise null
           if (response.item() != null && response.item().containsKey(mapName))
{
               return response.item().get(mapName).m();
           }

           return null;
       } catch (DynamoDbException e) {
           throw DynamoDbException.builder()
               .message("Failed to get map attribute: " + e.getMessage())
               .cause(e)
               .build();
       }
   }
```

Exemple d'utilisation d'opérations cartographiques avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating map operations in DynamoDB");

       try {
           // Example 1: Create a map with multiple attributes
```

```
            System.out.println("\nExample 1: Creating a map with multiple
attributes");
            Map<String, AttributeValue> productDetails = new HashMap<>();
            productDetails.put("Color",
AttributeValue.builder().s("Red").build());
            productDetails.put("Weight",
AttributeValue.builder().n("2.5").build());
            productDetails.put(
                "Dimensions", AttributeValue.builder().s("10x20x5").build());

            UpdateItemResponse createResponse =
                createMapWithAttributes(dynamoDbClient, tableName, key,
"Details", productDetails);

            System.out.println("Created map attribute: " +
createResponse.attributes());

            // Example 2: Update a specific attribute in a map
            System.out.println("\nExample 2: Updating a specific attribute in a
map");
            UpdateItemResponse updateResponse = updateMapAttributeSafe(
                dynamoDbClient,
                tableName,
                key,
                "Details",
                "Color",
                AttributeValue.builder().s("Blue").build());

            System.out.println("Updated map attribute: " +
updateResponse.attributes());

            // Example 3: Add an attribute to a nested map
            System.out.println("\nExample 3: Adding an attribute to a nested
map");
            UpdateItemResponse nestedResponse = addToNestedMap(
                dynamoDbClient,
                tableName,
                key,
                List.of("Specifications", "Technical", "Resolution"),
                AttributeValue.builder().s("1920x1080").build());

            System.out.println("Added to nested map: " +
nestedResponse.attributes());
```

```
            // Example 4: Remove an attribute from a map
            System.out.println("\nExample 4: Removing an attribute from a map");
            UpdateItemResponse removeResponse =
                removeMapAttribute(dynamoDbClient, tableName, key, "Details",
"Dimensions");

            System.out.println("Updated map after removal: " +
removeResponse.attributes());

            // Example 5: Get the current value of a map attribute
            System.out.println("\nExample 5: Getting the current value of a map
attribute");
            Map<String, AttributeValue> currentMap =
getMapAttribute(dynamoDbClient, tableName, key, "Details");

            if (currentMap != null) {
                System.out.println("Current map attribute:");
                for (Map.Entry<String, AttributeValue> entry :
currentMap.entrySet()) {
                    System.out.println("  " + entry.getKey() + ": " +
entry.getValue());
                }
            } else {
                System.out.println("Map attribute not found");
            }

            // Explain map operations
            System.out.println("\nKey points about DynamoDB map operations:");
            System.out.println("1. Maps are unordered collections of name-value
pairs");
            System.out.println("2. Use dot notation (map.key) to access or update
specific attributes");
            System.out.println("3. You can update individual attributes without
overwriting the entire map");
            System.out.println("4. Maps can be nested to create complex data
structures");
            System.out.println("5. Use REMOVE to delete attributes from a map");
            System.out.println("6. You can create a map with multiple attributes
in a single operation");
            System.out.println("7. Map keys are case-sensitive");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez les opérations cartographiques en utilisant AWS SDK pour JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update
 scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not
 exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
```

```
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update
 expression is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };
```

```
      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
```

```
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
```

```
      updateExpression += ` ${parentPath} =
 if_not_exists(${parentPath}, :emptyMap${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
```

```
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
```

```
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
```

```
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists
 (elegant approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);

    // Example 3: Update a deeply nested attribute
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );
```

```
    console.log("Updated multiple fields:", response4.Attributes);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds
 or fails");
    console.log("5. Using a single operation is more efficient than multiple
 separate updates");

  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez les opérations cartographiques en utilisant AWS SDK pour Python (Boto3).

```
"""
Example of updating map attributes in DynamoDB.

This module demonstrates how to update map attributes in DynamoDB, including
handling cases where the map attribute might not exist yet.
"""


import boto3
from typing import Any, Dict, Optional



def update_map_attribute_safe(
    table_name: str, key: Dict[str, Any], map_name: str, map_key: str, value: Any
) -> Dict[str, Any]:
    """
    Update a specific key in a map attribute, creating the map if it doesn't
 exist.

    This function demonstrates how to safely update a key within a map attribute,
    even if the map doesn't exist yet in the item.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use SET with attribute_not_exists to safely update the map
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map.#key = :value",
        ExpressionAttributeNames={"#map": map_name, "#key": map_key},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_nested_map(
    table_name: str, key: Dict[str, Any], path: str, value: Any
) -> Dict[str, Any]:
    """
    Add or update a value in a deeply nested map structure.

    This function demonstrates how to update a value at a specific path in a
    nested map structure, creating any intermediate maps as needed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested attribute (e.g.,
 "user.preferences.theme").
        value (Any): The value to set at the specified path.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}
```

```
    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_map_with_if_not_exists(
    table_name: str,
    key: Dict[str, Any],
    map_name: str,
    map_key: str,
    value: Any,
    default_map: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update a key in a map, creating the map with default values if it doesn't
 exist.

    This function demonstrates how to use if_not_exists to initialize a map with
    default values if it doesn't exist yet, and then update a specific key.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.
        default_map (Optional[Dict[str, Any]]): Default map values if the map
 doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Set default map if not provided
    if default_map is None:
        default_map = {}

    # Create a map with the new key-value pair
    updated_map = default_map.copy()
    updated_map[map_key] = value

    # Use if_not_exists to initialize the map if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map = if_not_exists(#map, :default_map)",
        ExpressionAttributeNames={"#map": map_name},
        ExpressionAttributeValues={":default_map": updated_map},
        ReturnValues="UPDATED_NEW",
    )

    return response


def merge_into_map(
    table_name: str, key: Dict[str, Any], map_name: str, values_to_merge:
 Dict[str, Any]
) -> Dict[str, Any]:
    """
    Merge multiple key-value pairs into a map attribute.

    This function demonstrates how to update multiple keys in a map attribute
    in a single operation, without overwriting the entire map.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        values_to_merge (Dict[str, Any]): Key-value pairs to merge into the map.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression for each key-value pair
    update_expression = "SET "
    expression_attribute_names = {"#map": map_name}
    expression_attribute_values = {}

    # Add each key-value pair to the update expression
    for i, (k, v) in enumerate(values_to_merge.items()):
        key_placeholder = f"#key{i}"
        value_placeholder = f":value{i}"

        expression_attribute_names[key_placeholder] = k
        expression_attribute_values[value_placeholder] = v

        if i > 0:
            update_expression += ", "
        update_expression += f"#map.{key_placeholder} = {value_placeholder}"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def example_usage():
    """Example of how to use the map attribute update functions."""
    # Example parameters
    table_name = "UserProfiles"
    key = {"UserId": "user123"}

    print("Example 1: Updating a specific key in a map attribute")
    try:
        response = update_map_attribute_safe(
            table_name=table_name, key=key, map_name="Preferences",
 map_key="Theme", value="Dark"
        )
        print(f"Map attribute updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating map attribute: {e}")

    print("\nExample 2: Adding a value to a deeply nested map")
    try:
        response = add_to_nested_map(
            table_name=table_name, key=key, path="Settings.Notifications.Email",
 value=True
        )
        print(f"Nested map updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating nested map: {e}")

    print("\nExample 3: Initializing a map with default values if it doesn't
 exist")
    try:
        default_map = {"Language": "English", "Currency": "USD", "TimeZone":
 "UTC"}

        response = update_map_with_if_not_exists(
            table_name=table_name,
            key={"UserId": "newuser456"},
            map_name="Preferences",
            map_key="Theme",
            value="Light",
            default_map=default_map,
        )
        print(f"Map initialized with defaults: {response.get('Attributes', {})}")
```

```
    except Exception as e:
        print(f"Error initializing map: {e}")

    print("\nExample 4: Merging multiple values into a map")
    try:
        values_to_merge = {
            "NotificationsEnabled": True,
            "EmailFrequency": "Daily",
            "PushNotifications": False,
        }

        response = merge_into_map(
            table_name=table_name,
            key=key,
            map_name="NotificationSettings",
            values_to_merge=values_to_merge,
        )
        print(f"Multiple values merged into map: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error merging values into map: {e}")

    print("\nBest practices for working with map attributes in DynamoDB:")
    print("1. Use dot notation to access and update nested attributes")
    print("2. Use ExpressionAttributeNames to handle reserved words and special
 characters")
    print("3. Use if_not_exists() to handle cases where attributes might not
 exist")
    print("4. Update specific map keys rather than overwriting the entire map")
    print("5. Use a single update operation to modify multiple map keys for
 better performance")
    print("6. Consider your data model carefully to minimize the need for deeply
 nested attributes")



if __name__ == "__main__":
    example_usage()
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Exécution d'opérations définies dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment exécuter des opérations d'ensemble dans DynamoDB.

- Ajout d'éléments à un attribut d'ensemble.
- Supprimez des éléments dans un attribut d'ensemble.
- Utilisez des opérations ADD et DELETE avec des ensembles.

Java

SDK pour Java 2.x

Démontrez les opérations du set en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

    /**
     * Adds values to a string set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a string set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToStringSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<String> valuesToAdd) {

        // Create a string set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ss(valuesToAdd).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds values to a number set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a number set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
```

```
     */
    public static UpdateItemResponse addToNumberSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<Number> valuesToAdd) {

        // Convert numbers to strings for DynamoDB
        Set<String> stringValues = new HashSet<>();
        for (Number value : valuesToAdd) {
            stringValues.add(value.toString());
        }

        // Create a number set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ns(stringValues).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes values from a set attribute.
     *
     * <p>This method demonstrates how to use the DELETE operation to remove
values
     * from a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToRemove The values to remove from the set
```

```
     * @param isNumberSet Whether the set is a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeFromSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> valuesToRemove,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("DELETE #setAttr :valuesToRemove")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
```

```
            .expressionAttributeValues(Map.of(":valuesToRemove", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Checks if a value exists in a set attribute.
     *
     * <p>This method demonstrates how to use the contains function to check
     * if a value exists in a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to check
     * @param setAttributeName The name of the set attribute
     * @param valueToCheck The value to check for
     * @return Map containing the result of the check
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> checkIfValueInSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        String valueToCheck) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters with a condition expression
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #tempAttr = :tempVal")
                .conditionExpression("contains(#setAttr, :valueToCheck)")
                .expressionAttributeNames(Map.of("#setAttr", setAttributeName,
"#tempAttr", "TempAttribute"))
                .expressionAttributeValues(Map.of(
                    ":valueToCheck",
AttributeValue.builder().s(valueToCheck).build(),
                    ":tempVal", AttributeValue.builder().s("TempValue").build()))
```

```
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Attempt the update operation
            dynamoDbClient.updateItem(request);

            // If we get here, the condition was met
            result.put("exists", true);
            result.put("message", "Value '" + valueToCheck + "' exists in the
set");

            // Clean up the temporary attribute
            UpdateItemRequest cleanupRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("REMOVE #tempAttr")
                .expressionAttributeNames(Map.of("#tempAttr", "TempAttribute"))
                .build();

            dynamoDbClient.updateItem(cleanupRequest);

        } catch (DynamoDbException e) {
            if (e.getMessage().contains("ConditionalCheckFailed")) {
                // The condition was not met
                result.put("exists", false);
                result.put("message", "Value '" + valueToCheck + "' does not
exist in the set");
            } else {
                // Some other error occurred
                result.put("exists", false);
                result.put("message", "Error checking set: " + e.getMessage());
                result.put("error", e.getClass().getSimpleName());
            }
        }

        return result;
    }

    /**
     * Creates a set with multiple values in a single operation.
     *
     * <p>This method demonstrates how to create a set with multiple values
     * in a single update operation.
     *
```

```
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param setValues The values to include in the set
     * @param isNumberSet Whether to create a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createSetWithValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> setValues,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a set attribute.
     *
     * <p>Helper method to retrieve the current value of a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param setAttributeName The name of the set attribute
     * @return The set attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static AttributeValue getSetAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String setAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(setAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the set attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(setAttributeName)) {
                return response.item().get(setAttributeName);
```

```
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get set attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemple d'utilisation d'opérations définies avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating set operations in DynamoDB");

        try {
            // Example 1: Create a string set with multiple values
            System.out.println("\nExample 1: Creating a string set with multiple
values");
            Set<String> tags = new HashSet<>();
            tags.add("Electronics");
            tags.add("Gadget");
            tags.add("Smartphone");

            UpdateItemResponse createResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Tags", tags, false // Not a
number set
                );

            System.out.println("Created set attribute: " +
createResponse.attributes());

            // Example 2: Add values to a string set
            System.out.println("\nExample 2: Adding values to a string set");
            Set<String> additionalTags = new HashSet<>();
            additionalTags.add("Mobile");
```

```
            additionalTags.add("Wireless");

            UpdateItemResponse addResponse = addToStringSet(dynamoDbClient,
tableName, key, "Tags", additionalTags);

            System.out.println("Updated set attribute: " +
addResponse.attributes());

            // Example 3: Create a number set with multiple values
            System.out.println("\nExample 3: Creating a number set with multiple
values");
            Set<Number> ratings = new HashSet<>();
            ratings.add(4);
            ratings.add(5);
            ratings.add(4.5);

            UpdateItemResponse createNumberSetResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Ratings", ratings, true // Is a
number set
                );

            System.out.println("Created number set attribute: " +
createNumberSetResponse.attributes());

            // Example 4: Add values to a number set
            System.out.println("\nExample 4: Adding values to a number set");
            Set<Number> additionalRatings = new HashSet<>();
            additionalRatings.add(3.5);
            additionalRatings.add(4.2);

            UpdateItemResponse addNumberResponse =
                addToNumberSet(dynamoDbClient, tableName, key, "Ratings",
additionalRatings);

            System.out.println("Updated number set attribute: " +
addNumberResponse.attributes());

            // Example 5: Remove values from a set
            System.out.println("\nExample 5: Removing values from a set");
            Set<String> tagsToRemove = new HashSet<>();
            tagsToRemove.add("Gadget");

            UpdateItemResponse removeResponse = removeFromSet(
```

```
                dynamoDbClient, tableName, key, "Tags", tagsToRemove, false // 
Not a number set
                );

            System.out.println("Updated set after removal: " + 
removeResponse.attributes());

            // Example 6: Check if a value exists in a set
            System.out.println("\nExample 6: Checking if a value exists in a 
set");
            Map<String, Object> checkResult = checkIfValueInSet(dynamoDbClient, 
tableName, key, "Tags", "Electronics");

            System.out.println("Check result: " + checkResult.get("message"));

            // Example 7: Get the current value of a set attribute
            System.out.println("\nExample 7: Getting the current value of a set 
attribute");
            AttributeValue currentStringSet = getSetAttribute(dynamoDbClient, 
tableName, key, "Tags");

            if (currentStringSet != null && currentStringSet.ss() != null) {
                System.out.println("Current string set values: " + 
currentStringSet.ss());
            } else {
                System.out.println("String set attribute not found");
            }

            AttributeValue currentNumberSet = getSetAttribute(dynamoDbClient, 
tableName, key, "Ratings");

            if (currentNumberSet != null && currentNumberSet.ns() != null) {
                System.out.println("Current number set values: " + 
currentNumberSet.ns());
            } else {
                System.out.println("Number set attribute not found");
            }

            // Explain set operations
            System.out.println("\nKey points about DynamoDB set operations:");
            System.out.println(
                "1. DynamoDB supports three set types: string sets (SS), number 
sets (NS), and binary sets (BS)");
```

```
            System.out.println("2. Sets can only contain elements of the same
 type");
            System.out.println("3. Use ADD to add elements to a set");
            System.out.println("4. Use DELETE to remove elements from a set");
            System.out.println("5. Sets automatically remove duplicate values");
            System.out.println("6. Sets are unordered collections");
            System.out.println("7. Use the contains function to check if a value
 exists in a set");
            System.out.println("8. You can create a set with multiple values in a
 single operation");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez les opérations du set en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
```

```
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}
```

```
/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from
 a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
```

```
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set
 attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
```

```
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
```

```
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
```

```
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
      [lastElement],
      setType
    );

    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations définies avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
```

```
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
      key,
      "Interests",
      ["Cooking"],
      "string"
    );

    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
```

```
    key,
    "FavoriteNumbers",
    [7, 42, 99],
    "number"
  );

  console.log("Created number set:", response4.Attributes);

  // Example 5: Replace an entire set
  console.log("\nExample 5: Replacing an entire set");
  const response5 = await replaceSet(
    config,
    tableName,
    key,
    "Interests",
    ["Gaming", "Movies", "Music"],
    "string"
  );

  console.log("Replaced set:", response5.Attributes);

  // Example 6: Remove the last element from a set
  console.log("\nExample 6: Removing the last element from a set");

  // First, create a set with just one element
  await createSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags",
    ["LastTag"],
    "string"
  );

  // Then, remove the last element
  const response6 = await removeLastElementFromSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags"
  );

  console.log(response6.message);
  console.log("Removed value:", response6.removedValue);
```

```
    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are
 automatically removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number
 sets, and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez les opérations du set en utilisant AWS SDK pour Python (Boto3).

```
import boto3
from typing import Any, Dict, List


def create_set_attribute(
```

```
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    set_values: List[Any],
    set_type: str = "string",
) -> Dict[str, Any]:
    """
    Create a new set attribute or add elements to an existing set.

    This function demonstrates how to use the ADD operation to create a new set
    or add elements to an existing set.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        set_values (List[Any]): The values to add to the set.
        set_type (str, optional): The type of set to create: "string", "number",
 or "binary".
            Defaults to "string".

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a DynamoDB set based on the specified type
    if set_type == "string":
        dynamo_set = set(str(value) for value in set_values)
    elif set_type == "number":
        # We need to use actual float values for the DynamoDB API
        # but mypy expects strings in sets, so we need to use type: ignore
        dynamo_set = set(float(value) for value in set_values)  # type: ignore
    else:  # binary set is not directly supported in high-level API, handled
 differently
        raise ValueError("Binary sets are not supported in this example")

    # Use the ADD operation to create or update the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :set_values",
```

```
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":set_values": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_add: List[Any]
) -> Dict[str, Any]:
    """
    Add elements to an existing set attribute.

    This function demonstrates how to use the ADD operation to add elements to an
 existing set.
    If the set doesn't exist, it will be created.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_add (List[Any]): The values to add to the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_add)

    # Use the ADD operation to add values to the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :values_to_add",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_add": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def remove_from_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_remove:
 List[Any]
) -> Dict[str, Any]:
    """
    Remove elements from a set attribute.

    This function demonstrates how to use the DELETE operation to remove elements
 from a set.
    If the last element is removed, the attribute will be deleted entirely.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_remove (List[Any]): The values to remove from the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_remove)

    # Use the DELETE operation to remove values from the set
    response = table.update_item(
        Key=key,
        UpdateExpression="DELETE #set_attr :values_to_remove",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_remove": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def check_if_set_exists(table_name: str, key: Dict[str, Any], set_name: str) ->
 bool:
    """
    Check if a set attribute exists in an item.

    This function demonstrates how to check if a set attribute exists after
    potentially removing all elements from it.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to check.
        set_name (str): The name of the set attribute.

    Returns:
        bool: True if the set attribute exists, False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Get the item
    response = table.get_item(
        Key=key, ProjectionExpression="#set_attr",
 ExpressionAttributeNames={"#set_attr": set_name}
    )

    # Check if the item exists and has the set attribute
    return "Item" in response and set_name in response["Item"]


def demonstrate_last_element_removal(
    table_name: str, key: Dict[str, Any], set_name: str
) -> Dict[str, Any]:
    """
    Demonstrate what happens when you remove the last element from a set.

    This function creates a set with a single element, then removes that element,
    showing that the attribute is completely removed when the last element is
 deleted.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
```

```
    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Step 1: Create a set with a single element
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=["last_element"],
        set_type="string",
    )

    # Step 2: Check that the set exists
    exists_before = check_if_set_exists(table_name, key, set_name)

    # Step 3: Remove the last element
    delete_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=["last_element"]
    )

    # Step 4: Check if the set still exists
    exists_after = check_if_set_exists(table_name, key, set_name)

    # Return the results
    return {
        "create_response": create_response,
        "exists_before": exists_before,
        "delete_response": delete_response,
        "exists_after": exists_after,
    }


def work_with_number_set(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    initial_values: List[float],
    values_to_add: List[float],
    values_to_remove: List[float],
) -> Dict[str, Any]:
    """
    Demonstrate working with a number set in DynamoDB.
```

```
    This function shows how to create and manipulate a set of numbers.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        initial_values (List[float]): The initial values for the set.
        values_to_add (List[float]): Values to add to the set.
        values_to_remove (List[float]): Values to remove from the set.

    Returns:
        Dict[str, Any]: A dictionary containing the responses from each
operation.
    """
    # Step 1: Create the number set
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=initial_values,
        set_type="number",
    )

    # Step 2: Add more numbers to the set
    add_response = add_to_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_add=values_to_add
    )

    # Step 3: Remove some numbers from the set
    remove_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_remove=values_to_remove
    )

    # Step 4: Get the final state
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    get_response = table.get_item(
        Key=key,
        ProjectionExpression=f"#{set_name}",
        ExpressionAttributeNames={f"#{set_name}": set_name},
```

```
    )

    # Return all responses
    return {
        "create_response": create_response,
        "add_response": add_response,
        "remove_response": remove_response,
        "final_state": get_response.get("Item", {}),
    }
```

Exemple d'utilisation d'opérations définies avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the set operations functions."""
    # Example parameters
    table_name = "UserPreferences"
    key = {"UserId": "user123"}

    print("Example 1: Creating a string set attribute")
    try:
        response = create_set_attribute(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            set_values=["AWS", "DynamoDB", "NoSQL"],
            set_type="string",
        )
        print(f"Set attribute created successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error creating set attribute: {e}")

    print("\nExample 2: Adding elements to an existing set")
    try:
        response = add_to_set(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            values_to_add=["Database", "Serverless"],
        )
```

```
        print(f"Elements added to set successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error adding to set: {e}")

    print("\nExample 3: Removing elements from a set")
    try:
        response = remove_from_set(
            table_name=table_name, key=key, set_name="FavoriteTags",
values_to_remove=["NoSQL"]
        )
        print(f"Elements removed from set successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error removing from set: {e}")

    print("\nExample 4: Demonstrating what happens when you remove the last
element from a set")
    try:
        results = demonstrate_last_element_removal(
            table_name=table_name, key={"UserId": "tempUser"},
set_name="SingleElementSet"
        )

        print(f"Set exists before removal: {results['exists_before']}")
        print(f"Set exists after removal: {results['exists_after']}")

        if not results["exists_after"]:
            print("The set attribute was completely removed when the last element
was deleted.")
        else:
            print("The set attribute still exists after removing the last
element.")
    except Exception as e:
        print(f"Error in last element removal demonstration: {e}")

    print("\nExample 5: Working with a number set")
    try:
        results = work_with_number_set(
            table_name=table_name,
            key={"UserId": "user456"},
            set_name="LuckyNumbers",
            initial_values=[7, 13, 42],
            values_to_add=[99, 100],
```

```
            values_to_remove=[13],
        )

        print(f"Initial number set: {results['create_response'].get('Attributes',
{})}")
        print(f"After adding numbers: {results['add_response'].get('Attributes',
{})}")
        print(f"After removing numbers:
{results['remove_response'].get('Attributes', {})}")
        print(f"Final state: {results['final_state']}")
    except Exception as e:
        print(f"Error working with number set: {e}")

    print("\nKey Points About DynamoDB Sets:")
    print("1. Sets can only contain elements of the same type (string, number, or
binary)")
    print("2. Sets automatically eliminate duplicate values")
    print("3. The ADD operation creates a set if it doesn't exist")
    print("4. The DELETE operation removes specified elements from a set")
    print("5. When the last element is removed from a set, the entire attribute
is deleted")
    print("6. Empty sets are not allowed in DynamoDB")
    print("7. Sets are unordered collections")
    print("8. The ADD operation is atomic for sets")
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB à l'aide de lots d'instructions PartiQL et d'un SDK AWS

Les exemples de code suivants montrent comment :

- Obtenez un lot d’éléments en exécutant plusieurs instructions SELECT.

- Ajoutez un lot d'éléments en exécutant plusieurs instructions INSERT.
- Mettez à jour un lot d'éléments en exécutant plusieurs instructions UPDATE.
- Supprimez un lot d'éléments en exécutant plusieurs instructions DELETE.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
```

```
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
```

```
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
```

```
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
```

```
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
```

```
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
```

```
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
```

```
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
```

```
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
```

```
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };
```

```
            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
```

```
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
              << std::endl;

    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
```

```
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
```

```
            return false;
        }
    }

    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;
```

```
        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
              << std::endl;

    // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
```

```
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }
```

```
        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
```

```
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
```

```
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
```

```
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario qui crée une table et exécute des lots de requêtes PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
```

```
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
```

```
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
log.Println("Updating a batch of movies with new ratings.")
err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
if err == nil {
 log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting projected data from the table to verify our update.")
log.Println("Using a page size of 2 to demonstrate paging.")
projections, err := runner.GetAllMovies(ctx, 2)
if err == nil {
 log.Println("All movies:")
 for _, projection := range projections {
  log.Println(projection)
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
```

```
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Créez une structure et des méthodes pouvant exécuter des instructions PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
```

```
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
```

```
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
```

```
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
```

```
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }
```

```
 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQBatch";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println("Creating an Amazon DynamoDB table named " + tableName
            + " with a key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Adding multiple records into the " + tableName
            + " table using a batch command.");
        putRecordBatch(ddb);
```

```
        // Update multiple movies by using the BatchExecute statement.
        String title1 = "Star Wars";
        int year1 = 1977;
        String title2 = "Wizard of Oz";
        int year2 = 1939;

        System.out.println("Query two movies.");
        getBatch(ddb, tableName, title1, title2, year1, year2);

        System.out.println("Updating multiple records using a batch command.");
        updateTableItemBatch(ddb);

        System.out.println("Deleting multiple records using a batch command.");
        deleteItemBatch(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static boolean getBatch(DynamoDbClient ddb, String tableName, String
title1, String title2, int year1, int year2) {
        String getBatch = "SELECT * FROM " + tableName + " WHERE title = ? AND
year = ?";

        List<BatchStatementRequest> statements = new ArrayList<>();
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title1).build(),
                AttributeValue.builder().n(String.valueOf(year1)).build())
            .build());
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title2).build(),
                AttributeValue.builder().n(String.valueOf(year2)).build())
            .build());

        BatchExecuteStatementRequest batchExecuteStatementRequest =
BatchExecuteStatementRequest.builder()
            .statements(statements)
            .build();

        try {
```

```
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchExecuteStatementRequest);
            if (!response.responses().isEmpty()) {
                response.responses().forEach(r -> {
                    System.out.println(r.item().get("title") + "\\t" +
r.item().get("year"));
                });
                return true;
            } else {
                System.out.println("Couldn't find either " + title1 + " or " +
title2 + ".");
                return false;
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();
```

```
        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse = dbWaiter
                .waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecordBatch(DynamoDbClient ddb) {
        String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            // Create three movies to add to the Amazon DynamoDB table.
            // Set data for Movie 1.
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n("1977")
                .build();
```

```
            AttributeValue att2 = AttributeValue.builder()
                .s("Star Wars")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parameters)
                .build();

            // Set data for Movie 2.
            List<AttributeValue> parametersMovie2 = new ArrayList<>();
            AttributeValue attMovie2 = AttributeValue.builder()
                .n("1939")
                .build();

            AttributeValue attMovie2A = AttributeValue.builder()
                .s("Wizard of Oz")
                .build();

            AttributeValue attMovie2B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie2.add(attMovie2);
            parametersMovie2.add(attMovie2A);
            parametersMovie2.add(attMovie2B);

            BatchStatementRequest statementRequestMovie2 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie2)
                .build();

            // Set data for Movie 3.
            List<AttributeValue> parametersMovie3 = new ArrayList<>();
```

```
            AttributeValue attMovie3 = AttributeValue.builder()
                .n(String.valueOf("2022"))
                .build();

            AttributeValue attMovie3A = AttributeValue.builder()
                .s("My Movie 3")
                .build();

            AttributeValue attMovie3B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie3.add(attMovie3);
            parametersMovie3.add(attMovie3A);
            parametersMovie3.add(attMovie3B);

            BatchStatementRequest statementRequestMovie3 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie3)
                .build();

            // Add all three movies to the list.
            List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
            myBatchStatementList.add(statementRequestMovie1);
            myBatchStatementList.add(statementRequestMovie2);
            myBatchStatementList.add(statementRequestMovie3);

            BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
                .statements(myBatchStatementList)
                .build();

            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Added new movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
    public static void updateTableItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "UPDATE MoviesPartiQBatch SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Update three records.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Update record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Update record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
```

```
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);
        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Updated three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    public static void deleteItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and
title=?";
```

```
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Specify three records to delete.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Specify record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Specify record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();
```

```
        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);

        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            ddb.batchExecuteStatement(batchRequest);
            System.out.println("Deleted three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }
}
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez des instructions PartiQL par lots.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
```

```
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
```

```
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");
```

```
  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
```

```
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
```

```
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}
```

```
suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
```

```
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
```

```
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
```

```
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
```

```
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Pour plus de détails sur l'API, consultez BatchExecuteStatementla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
```

```
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
```

```
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
```

```
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
```

```
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour PHP API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une classe capable d'exécuter des lots d'instructions PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
```

```
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
with
                           each statement. This list must be in the same order as
the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
            raise
        else:
            return output
```

Exécutez un scénario qui crée une table et exécute des requêtes ParIQL par lots.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
 database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
 mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
```

```
            "info": {
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)

    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
```

```
    output = wrapper.dyn_resource.meta.client.execute_statement(
        Statement=f'SELECT title, info.rating FROM "{table_name}"'
    )
    pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting a batch of movies from the table.")
    statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Pour plus de détails sur l'API, consultez BatchExecuteStatementle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario qui crée une table et exécute un lot de requêtes PartiQL.

```
 table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 sdk = DynamoDBPartiQLBatch.new(table_name)

 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Populate DynamoDB table with movie data.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(3, 'Select a batch of items from the movies table.')
 puts "Let's select some popular movies for side-by-side comparison."
 response = sdk.batch_execute_select([['Mean Girls', 2004], ['Goodfellas',
1977], ['The Prancing of the Lambs', 2005]])
 puts("Items selected: #{response['responses'].length}\n")
 print "\nDone!\n".green

 new_step(4, 'Delete a batch of items from the movies table.')
 sdk.batch_execute_write([['Mean Girls', 2004], ['Goodfellas', 1977], ['The
Prancing of the Lambs', 2005]])
 print "\nDone!\n".green
```

```
  new_step(5, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interroger une table DynamoDB à l'aide de PartiQL et d'un SDK AWS

Les exemples de code suivants montrent comment :

- Obtenez un élément en exécutant une instruction SELECT.
- Ajoutez un élément en exécutant une instruction INSERT.
- Mettez à jour un élément en exécutant une instruction UPDATE.
- Supprimez un élément en exécutant une instruction DELETE.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
```

```
        private static readonly AmazonDynamoDBClient Client = new
AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
```

```
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }
```

```
            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
```

```
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
```

```
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
```

```
   Aws::String title;
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                 << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                 << INFO_KEY << "': ?}";

       request.SetStatement(sqlStream.str());

       // Create the parameter attributes.
       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
       attributes.push_back(infoMapAttribute);
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
```

```
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();
```

```
            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
```

```
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
```

```
    }

    std::cout << "Deleting the movie" << std::endl;

    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;
```

```
    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
```

```
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
```

```
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
```

```
        count++;
    }
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario qui crée une table et exécute des requêtes PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
```

```
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config,
 tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovie := actions.Movie{
  Title: "24 Hour PartiQL People",
```

```
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

newRating := 6.6
log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
newRating)
err = runner.UpdateMovie(ctx, customMovie, newRating)
if err == nil {
 log.Printf("Updated %v with a new rating.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data again to verify the update.")
movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Deleting movie '%v'.\n", customMovie.Title)
err = runner.DeleteMovie(ctx, customMovie)
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
```

```
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Définissez une structure Movie utilisée dans cet exemple.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
```

```
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Créez une structure et des méthodes pouvant exécuter des instructions PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
```

```
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
```

```
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        String fileName = "../../../resources/sample_files/movies.json";
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(
            "******* Creating an Amazon DynamoDB table named MoviesPartiQ with a
 key named year and a sort key named title.");
```

```
        createTable(ddb, tableName);

        System.out.println("Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);

        System.out.println("Getting data from the MoviesPartiQ table.");
        getItem(ddb);

        System.out.println("Putting a record into the MoviesPartiQ table.");
        putRecord(ddb);

        System.out.println("Updating a record.");
        updateTableItem(ddb);

        System.out.println("Querying the movies released in 2013.");
        queryTable(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
```

```
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //Scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
```

```
        ObjectNode currentNode;
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf(year))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s(title)
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s(info)
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {
```

```
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n("2012")
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The Perks of Being a Wallflower")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecord(DynamoDbClient ddb) {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2020"))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("My Movie")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();
```

```
            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2013"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The East")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
 year";
```

```
        try {

            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();
            parameters.add(att1);

            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{

        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
executeStatementRequest(DynamoDbClient ddb, String statement,

List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
```

```
            .build();

        return ddb.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResponse
 executeStatementResult) {
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez des instructions PartiQL uniques.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */
```

```
  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)
```

```
    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }
```

```
    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))
```

```
        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
```

```
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- Pour plus de détails sur l'API, consultez ExecuteStatementla section AWS SDK pour la référence de l'API Kotlin.

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
```

```
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
```

```
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
```

```
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
```

```
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour PHP API.

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une classe capable d'exécuter des instructions PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint
```

```
import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
```

```
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

Exécutez un scénario qui crée une table et exécute des requêtes PartiQL.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
 stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
 'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)
```

```
    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
        [rating, title, year],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data again to verify our update.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
```

```
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Pour plus de détails sur l'API, consultez ExecuteStatementle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario qui crée une table et exécute des requêtes PartiQL.

```
table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLSingle.new(table_name)

new_step(1, 'Create a new DynamoDB table if none already exists.')
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, 'Populate DynamoDB table with movie data.')
download_file = 'moviedata.json'
puts("Downloading movie database to #{download_file}...")
movie_data = scaffold.fetch_movie_data(download_file)
puts("Writing movie data from #{download_file} into your table...")
scaffold.write_batch(movie_data)
puts("Records added: #{movie_data.length}.")
print "Done!\n".green
```

```
  new_step(3, 'Select a single item from the movies table.')
  response = sdk.select_item_by_title('Star Wars')
  puts("Items selected for title 'Star Wars': #{response.items.length}\n")
  print response.items.first.to_s.yellow
  print "\n\nDone!\n".green

  new_step(4, 'Update a single item from the movies table.')
  puts "Let's correct the rating on The Big Lebowski to 10.0."
  sdk.update_rating_by_title('The Big Lebowski', 1998, 10.0)
  print "\nDone!\n".green

  new_step(5, 'Delete a single item from the movies table.')
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title('The Silence of the Lambs', 1991)
  print "\nDone!\n".green

  new_step(6, 'Insert a new item into the movies table.')
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item('The Prancing of the Lambs', 2005, 'A movie about happy
 livestock.', 5.0)
  print "\nDone!\n".green

  new_step(7, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
```

```
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}
```

```
async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- Pour plus de détails sur l'API, voir ExecuteStatementla section de référence de l'API AWS SDK for Rust.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogation d'une table DynamoDB à l'aide d'un index secondaire global avec un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l’aide d’un index secondaire global.

- Interrogation d’une table DynamoDB à l’aide de sa clé primaire.
- Interrogez un index secondaires globaux (GSI) pour les autres modèles d’accès.
- Comparez les requêtes de table et les requêtes GSI.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB à l'aide de sa clé primaire et d'un index secondaire global (GSI) avec.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           System.out.println("Query on base table successful. Found " +
response.count() + " items");
           return response;
       } catch (ResourceNotFoundException e) {
           System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
           throw new DynamoDbQueryException("Table not found: " + tableName, e);
       } catch (DynamoDbException e) {
           System.err.println("Error querying base table: " + e.getMessage());
           throw new DynamoDbQueryException("Failed to execute query on base
table", e);
       }
   }

   /**
    * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
    *
    * @param tableName         The name of the DynamoDB table
    * @param indexName         The name of the GSI
    * @param partitionKeyName  The name of the GSI partition key attribute
    * @param partitionKeyValue The value of the GSI partition key to query
    * @return The query response from DynamoDB
    * @throws ResourceNotFoundException if the table or index doesn't exist
    * @throws DynamoDbException if the query fails
    */
   public QueryResponse queryGlobalSecondaryIndex(
       final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Index name", indexName);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Comparez l'interrogation directe d'une table à l'interrogation d'un GSI avec.AWS SDK for Java 2.x

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <basePartitionKeyName> <basePartitionKeyValue>
<gsiName> <gsiPartitionKeyName> <gsiPartitionKeyValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
```

```
                    basePartitionKeyName - The name of the base table partition
key attribute.
                    basePartitionKeyValue - The value of the base table partition
key to query.
                    gsiName - The name of the Global Secondary Index.
                    gsiPartitionKeyName - The name of the GSI partition key
attribute.
                    gsiPartitionKeyValue - The value of the GSI partition key to
query.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """";

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String basePartitionKeyName = args[1];
        final String basePartitionKeyValue = args[2];
        final String gsiName = args[3];
        final String gsiPartitionKeyName = args[4];
        final String gsiPartitionKeyValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        try (DynamoDbClient ddb =
DynamoDbClient.builder().region(region).build()) {
            final QueryTableAndGSI queryHelper = new QueryTableAndGSI(ddb);

            // Query the base table
            System.out.println("Querying base table where " +
basePartitionKeyName + " = " + basePartitionKeyValue);
            final QueryResponse tableResponse =
                queryHelper.queryTable(tableName, basePartitionKeyName,
basePartitionKeyValue);

            System.out.println("Found " + tableResponse.count() + " items in base
table:");
            tableResponse.items().forEach(item -> System.out.println(item));

            // Query the GSI
            System.out.println(
```

```
                "\nQuerying GSI '" + gsiName + "' where " + gsiPartitionKeyName +
" = " + gsiPartitionKeyValue);
            final QueryResponse gsiResponse =
                queryHelper.queryGlobalSecondaryIndex(tableName, gsiName,
gsiPartitionKeyName, gsiPartitionKeyValue);

            System.out.println("Found " + gsiResponse.count() + " items in
GSI:");
            gsiResponse.items().forEach(item -> System.out.println(item));

            // Explain the differences between querying a table and a GSI
            System.out.println("\nKey differences between querying a table and a
GSI:");
            System.out.println("1. When querying a GSI, you must specify the
indexName parameter");
            System.out.println("2. GSIs may not contain all attributes from the
base table (projection)");
            System.out.println("3. GSIs consume read capacity units from the
GSI's capacity, not the base table's");
            System.out.println("4. GSIs may have eventually consistent data
(cannot use ConsistentRead=true)");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table or index not found: " + e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB à l'aide de la clé primaire avec.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Interrogez un index secondaire global (GSI) DynamoDB avec.AWS SDK pour JavaScript

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB à l'aide de sa clé primaire et d'un index secondaire global (GSI) avec.AWS SDK pour Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB à l'aide d'une condition begins_with avec un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l'aide d'une condition begins_with.

- Utilisation de la fonction begins_with dans une expression de condition de clé.
- Filtrez des éléments en fonction d'un modèle de préfixe dans la clé de tri.

Java

SDK pour Java 2.x

Interrogation d'une table DynamoDB à l'aide d'une condition begins_with sur une clé de tri avec le kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithBeginsWithCondition(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String sortKeyName,
        final String sortKeyPrefix) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Sort key name", sortKeyName);
        CodeSampleUtils.validateStringParameter("Sort key prefix",
 sortKeyPrefix);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, sortKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_SK_PREFIX,
            AttributeValue.builder().s(sortKeyPrefix).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
```

```
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with begins_with condition successful.
Found {0} items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with begins_with condition",
e);
            throw e;
        }
    }
```

Démonstration de l'utilisation de begins_with avec différentes longueurs de préfixes à l'aide du kit AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        try {
            CodeSampleUtils.BeginsWithQueryConfig config =
CodeSampleUtils.BeginsWithQueryConfig.fromArgs(args);
            LOGGER.log(Level.INFO, "Querying items where {0} = {1} and {2} begins
with ''{3}''", new Object[] {
                config.getPartitionKeyName(),
                config.getPartitionKeyValue(),
                config.getSortKeyName(),
                config.getSortKeyPrefix()
            });

            // Using the builder pattern to create and execute the query
            final QueryResponse response = new BeginsWithQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withSortKeyName(config.getSortKeyName())
```

```
                .withSortKeyPrefix(config.getSortKeyPrefix())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> LOGGER.info(item.toString()));

            // Demonstrate with a different prefix
            if (!config.getSortKeyPrefix().isEmpty()) {
                String shorterPrefix = config.getSortKeyPrefix()
                    .substring(0, Math.max(1,
config.getSortKeyPrefix().length() / 2));
                LOGGER.log(Level.INFO, "\nNow querying with a shorter prefix:
''{0}''", shorterPrefix);

                final QueryResponse response2 = new BeginsWithQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withSortKeyName(config.getSortKeyName())
                    .withSortKeyPrefix(shorterPrefix)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with shorter prefix:",
response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }
        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            printUsage();
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found", e);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error", e);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error", e);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogation d'une table DynamoDB à l'aide d'une condition begins_with sur une clé de tri avec le kit AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
```

```
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB à l'aide d'une condition begins_with sur la clé de tri avec le kit AWS SDK pour Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_begins_with(
    table_name, partition_key_name, partition_key_value, sort_key_name, prefix
):
    """
    Query a DynamoDB table with a begins_with condition on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute.
```

```
        prefix (str): The prefix to match at the beginning of the sort key.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query with a begins_with condition on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) & Key(
        sort_key_name
    ).begins_with(prefix)
    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB en utilisant une plage de dates dans la clé de tri avec un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l'aide d'une plage de dates dans la clé de tri.

- Interrogation d'éléments dans une plage de dates spécifique.
- Utilisez des opérateurs de comparaison sur les clés de tri formatées par date.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB pour les éléments compris dans une plage de dates avec.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Montre comment interroger une table DynamoDB à l'aide d'un filtrage par plage de dates.

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<dateKeyName> <startDate> <endDate> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   dateKeyName - The name of the date attribute to filter on.
                   startDate - The start date for the range query (YYYY-MM-DD).
                   endDate - The end date for the range query (YYYY-MM-DD).
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 6) {
           System.out.println(usage);
           System.exit(1);
       }

       try {
           // Parse command line arguments into a config object
           CodeSampleUtils.DateRangeQueryConfig config =
CodeSampleUtils.DateRangeQueryConfig.fromArgs(args);

           LOGGER.log(
               Level.INFO, "Querying items from {0} to {1}", new Object[]
{config.getStartDate(), config.getEndDate()
               });

           // Using the builder pattern to create and execute the query
           final QueryResponse response = new DateRangeQueryBuilder()
               .withTableName(config.getTableName())
               .withPartitionKeyName(config.getPartitionKeyName())
               .withPartitionKeyValue(config.getPartitionKeyValue())
               .withDateKeyName(config.getDateKeyName())
               .withStartDate(config.getStartDate())
               .withEndDate(config.getEndDate())
               .withRegion(config.getRegion())
               .execute();
```

```
            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> {
                LOGGER.info(item.toString());

                // Extract and display the date attribute for clarity
                if (item.containsKey(config.getDateKeyName())) {
                    LOGGER.log(
                        Level.INFO,
                        "  Date attribute: {0}",
                        item.get(config.getDateKeyName()).s());
                }
            });

            // Demonstrate with a different date range
            LocalDate narrowerStartDate = config.getStartDate().plusDays(1);
            LocalDate narrowerEndDate = config.getEndDate().minusDays(1);

            if (!narrowerStartDate.isAfter(narrowerEndDate)) {
                LOGGER.log(Level.INFO, "\nNow querying with a narrower date
range: {0} to {1}", new Object[] {
                    narrowerStartDate, narrowerEndDate
                });

                final QueryResponse response2 = new DateRangeQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withDateKeyName(config.getDateKeyName())
                    .withStartDate(narrowerStartDate)
                    .withEndDate(narrowerEndDate)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with narrower date
range:", response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }

            LOGGER.info("\nNote: When storing dates in DynamoDB:");
            LOGGER.info("1. Use ISO format (YYYY-MM-DD) for lexicographical
ordering");
```

```
            LOGGER.info("2. Use the BETWEEN operator for inclusive date range
 queries");
            LOGGER.info("3. Consider using ISO-8601 format for timestamps with
 time components");

        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error: {0}", e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error: {0}", e.getMessage());
            System.exit(1);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB pour les éléments compris dans une plage de dates avec.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
```

```
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
```

```
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB pour les éléments compris dans une plage de dates avec.AWS SDK pour Python (Boto3)

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
```

```
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l'aide d'une expression de filtre complexe.

- Application d'expressions de filtre complexes aux résultats des requêtes.
- Combinez plusieurs conditions à l'aide d'opérateurs logiques.
- Filtrez des éléments en fonction d'attributs non essentiels.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {
```

```
        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());
```

```
        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide de.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
```

```
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide de.AWS SDK pour Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}
```

```
    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

```
def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
 max_price)
    rating_price_condition = None

    if min_rating is not None:
```

```
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB avec une expression de filtre dynamique à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l’aide d’une expression de filtre dynamique.

- Création d’expressions de filtre de manière dynamique lors de l’exécution.
- Construction de conditions de filtre en fonction des données saisies par l’utilisateur ou de l’état de l’application.
- Ajoutez ou supprimez des critères de filtre sous certaines conditions.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec une expression de filtre construite dynamiquement à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
```

```
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
e.getMessage());
            throw e;
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }
```

Montre comment utiliser des expressions de filtre dynamiques avec AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
```

```
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");
```

```
            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec une expression de filtre construite dynamiquement à l'aide de.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");
```

```
async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
```

```
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec une expression de filtre construite dynamiquement à l'aide de.AWS SDK pour Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
```

```
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
where
            keys are attribute names and values are dictionaries with 'operator'
and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
{'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
```

```
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Montre comment utiliser des expressions de filtre dynamiques avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_dynamic_filter function."""
    # Example parameters
    table_name = "Products"
    partition_key_name = "Category"
    partition_key_value = "Electronics"

    # Define dynamic filter conditions based on user input or runtime conditions
    user_min_rating = 4  # This could come from user input
    user_status_filter = "active"  # This could come from user input
```

```
    filter_conditions = {}

    # Only add conditions that are actually specified
    if user_min_rating is not None:
        filter_conditions["rating"] = {"operator": ">=", "value":
user_min_rating}

    if user_status_filter:
        filter_conditions["status"] = {"operator": "=", "value":
user_status_filter}

    print(
        f"Querying products in category '{partition_key_value}' with filter
conditions: {filter_conditions}"
    )

    # Execute the query with dynamic filter
    response = query_with_dynamic_filter(
        table_name, partition_key_name, partition_key_value, filter_conditions
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Product: {item}")
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB avec une expression de filtre et limitez avec un SDK AWS

Les exemples de code suivants montrent comment interroger une table à l'aide d'une expression de filtre et d'une limite.

- Application d'expressions de filtre aux résultats des requêtes en limitant le nombre d'éléments évalués.
- Comprenez comment la limite affecte les résultats de requête filtrés.
- Contrôlez le nombre maximum d'éléments traités dans une requête.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec une expression de filtre et limitez l'utilisation.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
```

```
        CodeSampleUtils.validateStringParameter("Filter attribute name",
filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
```

```
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
 e.getMessage());
            throw e;
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec une expression de filtre et limitez l'utilisation.AWS SDK pour Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
```

```
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
 filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

Montre comment utiliser des expressions de filtre contenant des limites AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_filter_and_limit function."""
    # Example parameters
    table_name = "ProductReviews"
    partition_key_name = "ProductId"
    partition_key_value = "P123456"
    filter_attribute = "Rating"
    filter_value = 3  # Filter for ratings > 3
    limit = 5
```

```
    print(f"Querying reviews for product '{partition_key_value}' with rating > 
{filter_value}")
    print(f"Limiting to {limit} evaluated items")

    # Execute the query with filter and limit
    response = query_with_filter_and_limit(
        table_name, partition_key_name, partition_key_value, filter_attribute, 
filter_value, limit
    )

    # Process the results
    items = response.get("Items", [])
    print(f"\nReturned {len(items)} items that passed the filter")

    for item in items:
        print(f"Review: {item}")

    # Explain the difference between Limit and actual results
    explain_limit_vs_results(response)

    # Check if there are more results
    if "LastEvaluatedKey" in response:
        print("\nThere are more results available. Use the LastEvaluatedKey for 
pagination.")
    else:
        print("\nAll matching results have been retrieved.")
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB avec des attributs imbriqués à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment interroger une table avec des attributs imbriqués.

- Accès aux éléments DynamoDB et filtrage en fonction des attributs imbriqués.
- Utilisez des expressions de chemin de document pour référencer des éléments imbriqués.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec des attributs imbriqués à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithNestedAttributes(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String nestedPath,
        final String nestedAttr,
        final String nestedValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Nested path", nestedPath);
        CodeSampleUtils.validateStringParameter("Nested attribute", nestedAttr);
        CodeSampleUtils.validateStringParameter("Nested value", nestedValue);

        // Split the nested path into components
```

```
        final String[] pathComponents = nestedPath.split("\\.");

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Build the nested attribute reference using document path notation
        final StringBuilder nestedAttributeRef = new StringBuilder();
        for (int i = 0; i < pathComponents.length; i++) {
            final String aliasName = "#n" + i;
            expressionAttributeNames.put(aliasName, pathComponents[i]);

            if (i > 0) {
                nestedAttributeRef.append(".");
            }
            nestedAttributeRef.append(aliasName);
        }

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_NESTED,
            AttributeValue.builder().s(nestedValue).build());

        // Create the filter expression using the nested attribute reference
        final String filterExpression = nestedAttributeRef + " = :nestedValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
```

```
            System.out.println("Query with nested attribute filter successful.
Found " + response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with nested attribute filter: " +
e.getMessage());
            throw e;
        }
    }
```

Montre comment interroger une table DynamoDB avec des attributs imbriqués.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<nestedPath> <nestedAttr> <nestedValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    nestedPath - The path to the nested map attribute (e.g.,
"address").
                    nestedAttr - The name of the nested attribute (e.g., "city").
                    nestedValue - The value to filter by (e.g., "Seattle").
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
```

```
        final String nestedPath = args[3];
        final String nestedAttr = args[4];
        final String nestedValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        System.out.println("Querying items where " + partitionKeyName + " = " +
partitionKeyValue + " and " + nestedPath
            + "." + nestedAttr + " = " + nestedValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new NestedAttributeQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withNestedPath(nestedPath)
                .withNestedAttribute(nestedAttr)
                .withNestedValue(nestedValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> {
                System.out.println(item);

                // Extract and display the nested attribute for clarity
                if (item.containsKey(nestedPath) && item.get(nestedPath).hasM())
{
                    Map<String, AttributeValue> nestedMap =
item.get(nestedPath).m();
                    if (nestedMap.containsKey(nestedAttr)) {
                        System.out.println("  Nested attribute " + nestedPath +
"." + nestedAttr + ": "
                                + formatAttributeValue(nestedMap.get(nestedAttr)));
                    }
                }
            });

            System.out.println("\nNote: When working with nested attributes in
DynamoDB:");
            System.out.println("1. Use dot notation in filter expressions to
access nested attributes");
```

```
            System.out.println("2. Use expression attribute names for each
 component of the path");
            System.out.println("3. Check if the nested attribute exists before
 accessing it");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec des attributs imbriqués à l'aide de.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec des attributs imbriqués à l'aide de.AWS SDK pour Python (Boto3)

```
from typing import Any, Dict, List
```

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_path: str,
    comparison_operator: str,
    comparison_value: Any,
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_path (str): The path to the nested attribute (e.g.,
 'specs.weight').
        comparison_operator (str): The comparison operator to use ('=', '!=',
 '<', '<=', '>', '>=').
        comparison_value (any): The value to compare against.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the filter expression based on the nested attribute path and
 comparison operator
    filter_expression = None
    if comparison_operator == "=":
        filter_expression = Attr(nested_path).eq(comparison_value)
    elif comparison_operator == "!=":
        filter_expression = Attr(nested_path).ne(comparison_value)
    elif comparison_operator == "<":
```

```
        filter_expression = Attr(nested_path).lt(comparison_value)
    elif comparison_operator == "<=":
        filter_expression = Attr(nested_path).lte(comparison_value)
    elif comparison_operator == ">":
        filter_expression = Attr(nested_path).gt(comparison_value)
    elif comparison_operator == ">=":
        filter_expression = Attr(nested_path).gte(comparison_value)
    elif comparison_operator == "contains":
        filter_expression = Attr(nested_path).contains(comparison_value)
    elif comparison_operator == "begins_with":
        filter_expression = Attr(nested_path).begins_with(comparison_value)

    # Execute the query with the filter expression
    response = table.query(KeyConditionExpression=key_condition,
 FilterExpression=filter_expression)

    return response


def query_with_multiple_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_conditions: List[Dict[str, Any]],
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by multiple nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_conditions (list): A list of dictionaries, each containing:
            - path (str): The path to the nested attribute
            - operator (str): The comparison operator
            - value (any): The value to compare against

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the combined filter expression for all nested attributes
    combined_filter = None

    for condition in nested_conditions:
        if not isinstance(condition, dict):
            continue
        path = condition.get("path", "")
        operator = condition.get("operator", "")
        value = condition.get("value")

        if not path or not operator:
            continue

        # Build the individual filter expression
        current_filter = None
        if operator == "=":
            current_filter = Attr(path).eq(value)
        elif operator == "!=":
            current_filter = Attr(path).ne(value)
        elif operator == "<":
            current_filter = Attr(path).lt(value)
        elif operator == "<=":
            current_filter = Attr(path).lte(value)
        elif operator == ">":
            current_filter = Attr(path).gt(value)
        elif operator == ">=":
            current_filter = Attr(path).gte(value)
        elif operator == "contains":
            current_filter = Attr(path).contains(value)
        elif operator == "begins_with":
            current_filter = Attr(path).begins_with(value)

        # Combine with the existing filter using AND
        if current_filter:
            if combined_filter is None:
                combined_filter = current_filter
            else:
                combined_filter = combined_filter & current_filter

    # Execute the query with the combined filter expression
```

```
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=combined_filter)

    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interroger une table DynamoDB avec pagination à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment interroger une table avec pagination.

- Mise en œuvre de la pagination pour les résultats des requêtes DynamoDB.
- Utilisez le LastEvaluatedKey pour récupérer les pages suivantes.
- Contrôlez le nombre d'éléments par page à l'aide du paramètre Limit.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec la pagination à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }
```

```
                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

Montre comment interroger une table DynamoDB avec pagination.

```
    public static void main(String[] args) {
```

```
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue> [pageSize]
[region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    pageSize (optional) - The maximum number of items to return
per page. (Default: 10)
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final int pageSize = args.length > 3 ? Integer.parseInt(args[3]) : 10;
        final Region region = args.length > 4 ? Region.of(args[4]) :
Region.US_EAST_1;

        System.out.println("Querying items with pagination (page size: " +
pageSize + ")");

        try {
            // Using the builder pattern to create and execute the query
            final List<Map<String, AttributeValue>> allItems = new
PaginationQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withPageSize(pageSize)
                .withRegion(region)
                .executeWithPagination();

            // Process the results
            System.out.println("\nSummary: Retrieved a total of " +
allItems.size() + " items");
```

```
            // Display the first few items as a sample
            final int sampleSize = Math.min(5, allItems.size());
            if (sampleSize > 0) {
                System.out.println("\nSample of retrieved items (first " +
sampleSize + "):");
                for (int i = 0; i < sampleSize; i++) {
                    System.out.println(allItems.get(i));
                }

                if (allItems.size() > sampleSize) {
                    System.out.println("... and " + (allItems.size() -
sampleSize) + " more items");
                }
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec la pagination à l'aide de.AWS SDK pour JavaScript

```
/**
```

```
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
```

```
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
```

```
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec la pagination à l'aide de.AWS SDK pour Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
```

```
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0
```

```
    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB avec des lectures très cohérentes à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment interroger une table avec des lectures fortement cohérentes.

- Configuration du niveau de cohérence pour les requêtes DynamoDB.
- Utilisez des lectures très cohérentes pour obtenir le maximum de up-to-date données.
- Compréhension des compromis entre une cohérence à terme et une forte cohérence.

Java

SDK pour Java 2.x

Interrogez une table DynamoDB avec une cohérence de lecture configurable à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec une cohérence de lecture configurable à l'aide de.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
```

```
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Interrogez une table DynamoDB avec l'option pour des lectures très cohérentes en utilisant.AWS SDK pour Python (Boto3)

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
```

```
        consistent_read (bool, optional): Whether to use strongly consistent
reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
ConsistentRead=consistent_read)

    return response
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez les données DynamoDB à l'aide des instructions PartiQL SELECT avec un SDK AWS

L'exemple de code suivant montre comment interroger des données à l'aide des instructions PartiQL SELECT.

JavaScript

SDK pour JavaScript (v3)

Interrogez des éléments d'une table DynamoDB à l'aide des instructions PartiQL SELECT avec.AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to query items from a DynamoDB table using
 PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
```

```
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };
```

```
  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ?
 AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456",
 "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01",
 "2023-12-31");
};
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez une table DynamoDB pour les éléments TTL à l'aide d'un SDK AWS

Les exemples de code suivants montrent l'interrogation d'éléments TTL.

Java

SDK pour Java 2.x

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK pour JavaScript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}
```

```
// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de.AWS SDK pour Python (Boto3)

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
```

```
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Interrogez les tables DynamoDB à l'aide de modèles de date et d'heure avec un SDK AWS

Les exemples de code suivants montrent comment interroger des tables à l'aide de modèles de date et d'heure.

- Stockez et interrogez date/time des valeurs dans DynamoDB.
- Mettez en œuvre des requêtes par plage de dates à l'aide de clés de tri.
- Formattez des chaînes de date pour une interrogation efficace.

Java

SDK pour Java 2.x

Requête utilisant des plages de dates dans les clés de tri avec AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
```

```
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Requête utilisant des variables date-heure avec.AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final String startDate,
        final String endDate) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateDateRangeParameters(dateKeyName, startDate,
 endDate);
        CodeSampleUtils.validateDateFormat("Start date", startDate);
        CodeSampleUtils.validateDateFormat("End date", endDate);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate", AttributeValue.builder().s(startDate).build());
        expressionAttributeValues.put(
```

```
            ":endDate", AttributeValue.builder().s(endDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with date range: " +
e.getMessage());
            throw e;
        }
    }
```

Exécution de requêtes dans des plages de dates dans les horodatages d'époque Unix à l'aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;
```

```
   public QueryResponse queryWithDateTimeEpoch(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final long startEpoch,
        final long endEpoch) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
       CodeSampleUtils.validateEpochTimestamp("Start epoch", startEpoch);
       CodeSampleUtils.validateEpochTimestamp("End epoch", endEpoch);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
       expressionAttributeNames.put("#dateKey", dateKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());
       expressionAttributeValues.put(
           ":startDate",
AttributeValue.builder().n(String.valueOf(startEpoch)).build());
       expressionAttributeValues.put(
           ":endDate",
AttributeValue.builder().n(String.valueOf(endEpoch)).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
```

```
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with epoch timestamps: " +
e.getMessage());
            throw e;
        }
    }
```

Effectuez des requêtes dans des plages de dates à l'aide LocalDateTime d'objets avec AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeLocalDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDateTime startDateTime,
        final LocalDateTime endDateTime) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        if (startDateTime == null || endDateTime == null) {
```

```
            throw new IllegalArgumentException("Start and end LocalDateTime must
 not be null");
        }

        // Convert LocalDateTime to ISO-8601 strings in UTC with the correct
 format
        final String startDate =
 startDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);
        final String endDate =
 endDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);

        return queryWithDateTime(tableName, partitionKeyName, partitionKeyValue,
 dateKeyName, startDate, endDate);
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for Java 2.x.

JavaScript

SDK pour JavaScript (v3)

Requête utilisant des plages de dates dans les clés de tri avec AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
```

```
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Requête utilisant des variables date-heure avec.AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
```

```
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript.

Python

Kit SDK for Python (Boto3)

Requête utilisant des plages de dates dans les clés de tri avec AWS SDK pour Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
```

```
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        sort_key_name (str): The name of the sort key attribute (containing date
values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

Requête utilisant des variables date-heure avec.AWS SDK pour Python (Boto3)

```
from datetime import datetime, timedelta
```

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_datetime(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range filter on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date/
time values).
        start_date (datetime): The start date/time for the query range.
        end_date (datetime): The end date/time for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date/time values as ISO 8601 strings
    # DynamoDB works well with ISO format for date/time values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
```

```
            ":end_date": end_date_str,
        },
    )

    return response



def example_usage():
    """Example of how to use the query_with_datetime function."""
    # Example parameters
    table_name = "Events"
    partition_key_name = "EventType"
    partition_key_value = "UserLogin"
    sort_key_name = "Timestamp"

    # Create date/time variables for the query
    end_date = datetime.now()
    start_date = end_date - timedelta(days=7)  # Query events from the last 7
 days

    print(f"Querying events from {start_date.isoformat()} to
 {end_date.isoformat()}")

    # Execute the query
    response = query_with_datetime(
        table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Event: {item}")
```

- Pour de plus amples informations sur l'API, consultez Requête dans la référence d'API AWS du kit SDK pour Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Enregistrez les informations EXIF et autres images à l'aide d'un SDK AWS

L'exemple de code suivant illustre comment :

- Obtenir des informations EXIF à partir d'un fichier JPG, JPEG ou PNG.
- Charger le fichier image sur un compartiment Amazon S3.
- Utiliser Amazon Rekognition pour identifier les trois principaux attributs (étiquettes) dans le fichier.
- Ajouter les informations EXIF et les étiquettes à un tableau Amazon DynamoDB dans la région.

Rust

SDK pour Rust

Obtenez les informations EXIF à partir d'un fichier JPG, JPEG ou PNG, chargez le fichier image dans un compartiment Amazon S3, utilisez Amazon Rekognition pour identifier les trois principaux attributs (étiquettes dans Amazon Rekognition) du fichier et ajoutez les informations EXIF et d'étiquettes à un tableau Amazon DynamoDB dans la région.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon Rekognition
- Amazon S3

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Configuration du contrôle d'accès basé sur les attributs pour DynamoDB à l'aide de la version 2 AWS Command Line Interface

L’exemple de code suivant montre comment mettre en place un Contrôle d’accès par attributs (ABAC) pour DynamoDB.

- Créez une politique IAM pour ABAC.
- Créez des tables avec des balises pour différents départements.
- Répertoriez et filtrez les tables en fonction des balises.

Bash

AWS CLI avec le script Bash

Créez une politique IAM pour ABAC.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
```

```
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
    --policy-document file://abac-policy.json
```

Créez des tables avec des balises pour différents départements.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
```

```
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

Répertoriez et filtrez les tables en fonction des balises.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreatePolicy
  - CreateTable
  - ListTables

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Comprendre l'ordre des expressions de mise à jour dans DynamoDB à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment comprendre l'ordre des expressions de mise à jour.

- Découverte de la façon dont DynamoDB traite les expressions de mise à jour.
- Comprenez l'ordre des opérations dans les expressions de mise à jour.
- Évitez les résultats inattendus en comprenant l'évaluation des expressions.

Java

SDK pour Java 2.x

Démontrez l'ordre des expressions de mise à jour en utilisant AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Demonstrates the effect of update expression order.
     *
     * <p>This method shows how the order of operations in an update expression
     * affects the result of the update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
```

```
     * @return Map containing the results of different update orders
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> demonstrateUpdateOrder(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Initialize the item with a counter
            UpdateItemRequest initRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :zero, OldCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            dynamoDbClient.updateItem(initRequest);

            // Example 1: SET first, then ADD
            UpdateItemRequest setFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value ADD
OldCounter :increment")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse setFirstResponse =
dynamoDbClient.updateItem(setFirstRequest);
            results.put("setFirstResponse", setFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 2: ADD first, then SET
            UpdateItemRequest addFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
```

```
                .key(key)
                .updateExpression("ADD Counter :increment SET OldCounter
= :value")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse addFirstResponse =
dynamoDbClient.updateItem(addFirstRequest);
            results.put("addFirstResponse", addFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 3: SET with multiple attributes
            UpdateItemRequest multiSetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value, OldCounter = Counter")
                .expressionAttributeValues(
                    Map.of(":value", AttributeValue.builder().n("10").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse multiSetResponse =
dynamoDbClient.updateItem(multiSetRequest);
            results.put("multiSetResponse", multiSetResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 4: SET with expression using the same attribute
            UpdateItemRequest selfReferenceRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = Counter + :increment, OldCounter
= Counter")
                .expressionAttributeValues(
                    Map.of(":increment",
AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();
```

```
            UpdateItemResponse selfReferenceResponse =
dynamoDbClient.updateItem(selfReferenceRequest);
            results.put("selfReferenceResponse", selfReferenceResponse);

            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Updates an item with SET first, then REMOVE.
     *
     * <p>This method demonstrates updating an item with SET operation first,
     * followed by a REMOVE operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithSetFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue REMOVE #removeAttr")
            .expressionAttributeNames(Map.of(
```

```
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with SET first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with REMOVE first, then SET.
     *
     * <p>This method demonstrates updating an item with REMOVE operation first,
     * followed by a SET operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithRemoveFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
```

```
            .key(key)
            .updateExpression("REMOVE #removeAttr SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with REMOVE first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with all operation types in a specific order.
     *
     * <p>This method demonstrates using all operation types (SET, REMOVE, ADD,
DELETE)
     * in a specific order in a single update expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithAllOperationTypes(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #stringAttr = :stringVal, #mapAttr.#nestedAttr
= :nestedVal " + "REMOVE #oldAttr "
```

```
                + "ADD #counterAttr :increment "
                + "DELETE #stringSetAttr :stringSetVal")
            .expressionAttributeNames(Map.of(
                "#stringAttr", "StringAttribute",
                "#mapAttr", "MapAttribute",
                "#nestedAttr", "NestedAttribute",
                "#oldAttr", "OldAttribute",
                "#counterAttr", "CounterAttribute",
                "#stringSetAttr", "StringSetAttribute"))
            .expressionAttributeValues(Map.of(
                ":stringVal", AttributeValue.builder().s("New Value").build(),
                ":nestedVal", AttributeValue.builder().s("Nested Value").build(),
                ":increment", AttributeValue.builder().n("1").build(),
                ":stringSetVal", AttributeValue.builder().ss("Value1").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with all operation types: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Gets the current state of an item.
     *
     * <p>Helper method to retrieve the current state of an item.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @return The item or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getItem(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {
```

```
        // Define the get parameters
        GetItemRequest request =
            GetItemRequest.builder().tableName(tableName).key(key).build();

        // Perform the get operation
        try {
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the item if it exists, otherwise null
            return response.item();
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get item: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemple d'utilisation de l'ordre des expressions de mise à jour avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating update expression order in DynamoDB");

        try {
            // Example 1: Demonstrate update order effects
            System.out.println("\nExample 1: Demonstrating update order
effects");
            Map<String, Object> orderResults =
demonstrateUpdateOrder(dynamoDbClient, tableName, key);

            if ((boolean) orderResults.get("success")) {
                System.out.println("SET first, then ADD:");
                System.out.println("  " + orderResults.get("setFirstResponse"));

                System.out.println("ADD first, then SET:");
                System.out.println("  " + orderResults.get("addFirstResponse"));
```

```
                System.out.println("SET with multiple attributes:");
                System.out.println("  " + orderResults.get("multiSetResponse"));

                System.out.println("SET with self-reference:");
                System.out.println("  " +
orderResults.get("selfReferenceResponse"));
            } else {
                System.out.println("Error: " + orderResults.get("error"));
            }

            // Example 2: Update with SET first, then REMOVE
            System.out.println("\nExample 2: Update with SET first, then
REMOVE");
            UpdateItemResponse setFirstResponse = updateWithSetFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Active").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
setFirstResponse.attributes());

            // Example 3: Update with REMOVE first, then SET
            System.out.println("\nExample 3: Update with REMOVE first, then
SET");
            UpdateItemResponse removeFirstResponse = updateWithRemoveFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Inactive").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
removeFirstResponse.attributes());

            // Example 4: Update with all operation types
            System.out.println("\nExample 4: Update with all operation types");
            UpdateItemResponse allOpsResponse =
updateWithAllOperationTypes(dynamoDbClient, tableName, key);
```

```
            System.out.println("Updated attributes: " +
allOpsResponse.attributes());

            // Example 5: Get the current state of the item
            System.out.println("\nExample 5: Current state of the item");
            Map<String, AttributeValue> item = getItem(dynamoDbClient, tableName,
key);

            if (item != null) {
                System.out.println("Item: " + item);
            } else {
                System.out.println("Item not found");
            }

            // Explain update expression order
            System.out.println("\nKey points about update expression order in
DynamoDB:");
            System.out.println("1. Update expressions are processed in this
order: SET, REMOVE, ADD, DELETE");
            System.out.println("2. Within each clause, operations are processed
from left to right");
            System.out.println("3. SET operations use the item state before any
updates in the expression");
            System.out.println("4. When an attribute is referenced multiple
times, the first operation wins");
            System.out.println("5. To reference a new value, split the update
into multiple operations");
            System.out.println("6. The order of clauses in the expression doesn't
change the evaluation order");
            System.out.println("7. For complex updates, consider using multiple
separate update operations");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez l'ordre des expressions de mise à jour en utilisant AWS SDK pour JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
```

```
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before
 modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
the value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
```

```
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
```

```
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
```

```
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a
 value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
```

```
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
    }
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));
```

```
  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple operations on the same attributes
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `
      SET counter = counter + :inc1,
          counter = counter + :inc2,
          map_attr.nested1 = :new_val1,
          map_attr.nested3 = :new_val3,
          list_attr[0] = list_attr[1],
          list_attr[1] = list_attr[2]
    `,
    ExpressionAttributeValues: {
      ":inc1": 5,
      ":inc2": 3,
      ":new_val1": "updated_value1",
      ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Exemple d'utilisation de l'ordre des expressions de mise à jour avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update
 expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null,
 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
    console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
 following:");
    console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
    console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
    console.log("   - Removed the attribute 'a'");
    console.log("3. The final state has b=1, c=2, and 'a' is removed");
    console.log("4. This demonstrates that DynamoDB uses the values of attributes
 as they were BEFORE any modifications");

    // Example 2: Demonstrating the order of different action types
    console.log("\nExample 2: Demonstrating the order of different action
 types");
    const results2 = await demonstrateActionOrder(
      config,
      tableName,
```

```
      { OrderId: "order456" }
    );

    console.log("Initial state:", JSON.stringify(results2.initialState, null,
2));
    console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
    console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The update expression contained multiple action types:
REMOVE, SET, ADD, DELETE");
    console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
    console.log("3. First, 'to_remove' was removed");
    console.log("4. Then, 'to_modify' was set to a new value");
    console.log("5. Next, 'counter' was incremented by 5");
    console.log("6. Finally, 'B' was removed from the set attribute");

    // Example 3: Updating multiple attributes in a single SET action
    console.log("\nExample 3: Updating multiple attributes in a single SET
action");
    const response3 = await updateWithMultipleSetActions(
      config,
      tableName,
      { OrderId: "order789" },
      {
        Status: "Shipped",
        ShippingDate: "2025-05-28",
        TrackingNumber: "1Z999AA10123456784"
      }
    );

    console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

    // Example 4: Conditional value copying with if_not_exists
    console.log("\nExample 4: Conditional value copying with if_not_exists");
    const response4 = await updateWithConditionalValueCopying(
      config,
      tableName,
      { OrderId: "order101" },
      "PreferredShippingMethod",
      "ShippingMethod",
```

```
      "Standard"
    );

    console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

    // Example 5: Multiple operations on the same attribute
    console.log("\nExample 5: Multiple operations on the same attribute");
    const results5 = await demonstrateMultipleOperationsOnSameAttribute(
      config,
      tableName,
      { OrderId: "order202" }
    );

    console.log("Initial state:", JSON.stringify(results5.initialState, null,
2));
    console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
    console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The counter was incremented twice (first by 5, then by 3) for
a total of +8");
    console.log("2. The map attribute had one value updated and a new nested
attribute added");
    console.log("3. The list attribute had values shifted (value at index 1 moved
to index 0, value at index 2 moved to index 1)");
    console.log("4. All operations within the SET action are processed from left
to right");

    // Key points about update expression order of operations
    console.log("\nKey Points About Update Expression Order of Operations:");
    console.log("1. Variables in expressions hold copies of attribute values as
they existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in
this order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left
to right");
    console.log("4. You can reference the same attribute multiple times in an
expression");
    console.log("5. You can use if_not_exists() to conditionally set values based
on attribute existence");
    console.log("6. Using a single update expression with multiple actions is
more efficient than multiple separate updates");
```

```
    console.log("7. The update expression is atomic - either all actions succeed
 or none do");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez l'ordre des expressions de mise à jour en utilisant AWS SDK pour Python (Boto3)

```
import boto3
import json
from typing import Any, Dict, Optional


def update_with_multiple_actions(
    table_name: str,
    key: Dict[str, Any],
    update_expression: str,
    expression_attribute_names: Optional[Dict[str, str]] = None,
    expression_attribute_values: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update an item with multiple actions in a single update expression.

    This function demonstrates how to use multiple actions in a single update
 expression
    and how DynamoDB processes these actions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        update_expression (str): The update expression with multiple actions.
        expression_attribute_names (Optional[Dict[str, str]]): Expression
 attribute name placeholders.
```

```
        expression_attribute_values (Optional[Dict[str, Any]]): Expression
 attribute value placeholders.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Prepare the update parameters
    update_params = {
        "Key": key,
        "UpdateExpression": update_expression,
        "ReturnValues": "UPDATED_NEW",
    }

    # Add expression attribute names if provided
    if expression_attribute_names:
        update_params["ExpressionAttributeNames"] = expression_attribute_names

    # Add expression attribute values if provided
    if expression_attribute_values:
        update_params["ExpressionAttributeValues"] = expression_attribute_values

    # Execute the update
    response = table.update_item(**update_params)

    return response


def demonstrate_value_copying(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate that variables hold copies of existing values before
 modifications.

    This function creates an item with initial values, then updates it with an
 expression
    that uses the values of attributes before they are modified in the same
 expression.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update({"a": 1, "b": 2, "c": 3})

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with an expression that uses values before they are
 modified
    # This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
    update_response = table.update_item(
        Key=key, UpdateExpression="REMOVE a SET b = a, c = b",
 ReturnValues="UPDATED_NEW"
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def demonstrate_action_order(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
```

```
    Demonstrate the order in which different action types are processed.

    This function creates an item with initial values, then updates it with an
expression
    that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
order
    in which they are processed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update(
        {
            "counter": 10,
            "set_attr": set(["A", "B", "C"]),
            "to_remove": "This will be removed",
            "to_modify": "Original value",
        }
    )

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with multiple action types
    # The actions will be processed in this order: REMOVE, SET, ADD, DELETE
    update_response = table.update_item(
        Key=key,
        UpdateExpression="REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
        ExpressionAttributeValues={
            ":new_value": "Updated value",
```

```
            ":increment": 5,
            ":elements": set(["B"]),
        },
        ReturnValues="UPDATED_NEW",
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def update_with_multiple_set_actions(
    table_name: str, key: Dict[str, Any], attributes: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Update multiple attributes with a single SET action.

    This function demonstrates how to update multiple attributes in a single SET
 action,
    which is more efficient than using multiple separate update operations.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attributes (Dict[str, Any]): The attributes to update and their new
 values.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression and expression attribute values
    update_expression = "SET "
```

```
    expression_attribute_values = {}

    # Add each attribute to the update expression
    for i, (attr_name, attr_value) in enumerate(attributes.items()):
        value_placeholder = f":val{i}"

        if i > 0:
            update_expression += ", "
        update_expression += f"{attr_name} = {value_placeholder}"

        expression_attribute_values[value_placeholder] = attr_value

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_with_conditional_value_copying(
    table_name: str,
    key: Dict[str, Any],
    source_attribute: str,
    target_attribute: str,
    default_value: Any,
) -> Dict[str, Any]:
    """
    Update an attribute with a value from another attribute or a default value.

    This function demonstrates how to use if_not_exists to conditionally copy a
 value
    from one attribute to another, or use a default value if the source doesn't
 exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        source_attribute (str): The attribute to copy the value from.
        target_attribute (str): The attribute to update.
```

```
        default_value (Any): The default value to use if the source attribute
doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to conditionally copy the value
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {target_attribute} =
if_not_exists({source_attribute}, :default)",
        ExpressionAttributeValues={":default": default_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Exemple d'utilisation de l'ordre des expressions de mise à jour avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use update expression order of operations in
DynamoDB."""
    # Example parameters
    table_name = "OrderProcessing"
    key = {"OrderId": "order123"}

    print("Example 1: Demonstrating value copying in update expressions")
    try:
        results = demonstrate_value_copying(table_name=table_name, key=key)

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
```

```
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The initial state had a=1, b=2, c=3")
        print("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:")
        print("   - Copied the value of 'a' (which was 1) to be used for 'b'")
        print("   - Copied the value of 'b' (which was 2) to be used for 'c'")
        print("   - Removed the attribute 'a'")
        print("3. The final state has b=1, c=2, and 'a' is removed")
        print(
            "4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications"
        )
    except Exception as e:
        print(f"Error demonstrating value copying: {e}")

    print("\nExample 2: Demonstrating the order of different action types")
    try:
        results = demonstrate_action_order(table_name=table_name, key={"OrderId":
"order456"})

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE")
        print("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE")
        print("3. First, 'to_remove' was removed")
        print("4. Then, 'to_modify' was set to a new value")
        print("5. Next, 'counter' was incremented by 5")
        print("6. Finally, 'B' was removed from the set attribute")
    except Exception as e:
        print(f"Error demonstrating action order: {e}")

    print("\nExample 3: Updating multiple attributes in a single SET action")
    try:
        response = update_with_multiple_set_actions(
            table_name=table_name,
```

```
            key={"OrderId": "order789"},
            attributes={
                "Status": "Shipped",
                "ShippingDate": "2025-05-14",
                "TrackingNumber": "1Z999AA10123456784",
            },
        )

        print(
            f"Multiple attributes updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating multiple attributes: {e}")

    print("\nExample 4: Conditional value copying with if_not_exists")
    try:
        response = update_with_conditional_value_copying(
            table_name=table_name,
            key={"OrderId": "order101"},
            source_attribute="PreferredShippingMethod",
            target_attribute="ShippingMethod",
            default_value="Standard",
        )

        print(
            f"Conditional value copying result:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error with conditional value copying: {e}")

    print("\nKey Points About Update Expression Order of Operations:")
    print(
        "1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications"
    )
    print(
        "2. Multiple actions in an update expression are processed in this order:
REMOVE, SET, ADD, DELETE"
    )
    print("3. Within each action type, operations are processed from left to
right")
```

```
    print("4. You can reference the same attribute multiple times in an
expression")
    print("5. You can use if_not_exists() to conditionally set values based on
attribute existence")
    print(
        "6. Using a single update expression with multiple actions is more
efficient than multiple separate updates"
    )
    print("7. The update expression is atomic - either all actions succeed or
none do")
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Mettre à jour un paramètre de table DynamoDB avec un débit chaud à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment mettre à jour le paramètre de débit chaud d'une table.

Java

SDK pour Java 2.x

Mise à jour du paramètre de débit chaud sur une table DynamoDB existante à l'aide du kit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
```

```
   public static WarmThroughput buildWarmThroughput(final Long
readUnitsPerSecond, final Long writeUnitsPerSecond) {
       return WarmThroughput.builder()
           .readUnitsPerSecond(readUnitsPerSecond)
           .writeUnitsPerSecond(writeUnitsPerSecond)
           .build();
   }

   /**
    * Updates a DynamoDB table with warm throughput settings for both the table
and a global secondary index.
    *
    * @param ddb The DynamoDB client
    * @param tableName The name of the table to update
    * @param tableReadUnitsPerSecond Read units per second for the table
    * @param tableWriteUnitsPerSecond Write units per second for the table
    * @param globalSecondaryIndexName The name of the global secondary index to
update
    * @param globalSecondaryIndexReadUnitsPerSecond Read units per second for
the GSI
    * @param globalSecondaryIndexWriteUnitsPerSecond Write units per second for
the GSI
    */
   public static void updateDynamoDBTable(
       final DynamoDbClient ddb,
       final String tableName,
       final Long tableReadUnitsPerSecond,
       final Long tableWriteUnitsPerSecond,
       final String globalSecondaryIndexName,
       final Long globalSecondaryIndexReadUnitsPerSecond,
       final Long globalSecondaryIndexWriteUnitsPerSecond) {

       final WarmThroughput tableWarmThroughput =
           buildWarmThroughput(tableReadUnitsPerSecond,
tableWriteUnitsPerSecond);
       final WarmThroughput gsiWarmThroughput =
           buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
globalSecondaryIndexWriteUnitsPerSecond);

       final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
GlobalSecondaryIndexUpdate.builder()
           .update(UpdateGlobalSecondaryIndexAction.builder()
               .indexName(globalSecondaryIndexName)
```

```
                .warmThroughput(gsiWarmThroughput)
                .build())
            .build();

        final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

        try {
            ddb.updateTable(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }

        System.out.println(SUCCESS_MESSAGE);
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Mettez à jour le paramètre de débit chaud sur une table DynamoDB existante à l'aide du kit AWS SDK pour JavaScript.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
```

```
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Mettez à jour le paramètre de débit chaud sur une table DynamoDB existante à l'aide du kit AWS SDK pour Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def update_dynamodb_table_warm_throughput(
    table_name,
    table_read_units,
    table_write_units,
    gsi_name,
    gsi_read_units,
    gsi_write_units,
    region_name="us-east-1",
):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    :return: The response from the update_table operation
    """
    try:
```

```
        ddb = client("dynamodb", region_name=region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units,
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units,
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {"Update": {"IndexName": gsi_name, "WarmThroughput":
gsi_warm_throughput}}
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput,
        }

        # Update the table
        response = ddb.update_table(**update_table_request)
        print("Table updated successfully!")
        return response  # Make sure to return the response
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

- Pour plus de détails sur l'API, consultez UpdateTablele AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Mettre à jour un élément DynamoDB avec un TTL à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment mettre à jour la TTL d’un élément.

Java

SDK pour Java 2.x

Mise à jour de la TTL sur un élément DynamoDB existant dans une table.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
```

```
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
```

```
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
```

```
)
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Mettre à jour les données DynamoDB à l'aide des instructions PARTIQL UPDATE avec un SDK AWS

L'exemple de code suivant montre comment mettre à jour des données à l'aide des instructions PartiQL UPDATE.

JavaScript

SDK pour JavaScript (v3)

Mettez à jour les éléments d'une table DynamoDB à l'aide des instructions PARTIQL UPDATE avec.AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to update items in a DynamoDB table using
 PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
```

```
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new
 values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
```

```
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName}
 = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
```

```
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
```

```
    "prod789",
    "quantity",
    5
  );

  // Update with a condition
  await updateItemWithCondition(
    "UsersTable",
    "userId",
    "user123",
    "userStatus",
    "active",
    "userType",
    "premium"
  );

  // Batch update multiple items
  await batchUpdateItems("UsersTable", [
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user123",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user456",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser API Gateway pour invoquer une fonction Lambda

Les exemples de code suivants montrent comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

Java

SDK pour Java 2.x

Montre comment créer une AWS Lambda fonction à l'aide de l'API d'exécution Lambda Java. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d’anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d’anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK pour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide de l'API JavaScript d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d’anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d’anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

Kit SDK for Python (Boto3)

Cet exemple montre comment créer et utiliser une API REST Amazon API Gateway qui cible une fonction AWS Lambda. Le gestionnaire Lambda explique comment router en fonction des méthodes HTTP, comment obtenir des données à partir de la chaîne de requête, de l'en-tête et du corps, et comment renvoyer une réponse JSON.

- Déployer une fonction Lambda.
- Créer une API REST avec API Gateway.
- Créer une ressource REST qui cible la fonction Lambda.
- Accorder à API Gateway l'autorisation d'invoquer la fonction Lambda.
- Utiliser le package Requests (Requêtes) pour envoyer des requêtes à l'API REST.
- Nettoyer toutes les ressources créées lors de la démonstration.

Il est préférable de visionner cet exemple sur GitHub. Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda

L'exemple de code suivant montre comment créer une machine àAWS Step Functionsétats qui invoque des AWS Lambda fonctions en séquence.

Java

SDK pour Java 2.x

Montre comment créer un flux de travail AWS sans serveur en utilisant AWS Step Functions et le AWS SDK for Java 2.x. Chaque étape du flux de travail est implémentée à l'aide d'une AWS Lambda fonction.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser un modèle de document pour DynamoDB à l'aide d'un SDK AWS

L'exemple de code suivant montre comment effectuer des opérations de création, de lecture, de mise à jour et de suppression (CRUD) et par lots à l'aide d'un modèle de document pour DynamoDB et d'un SDK.AWS

Pour en savoir plus, consultez la section Modèle de document.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Effectuez des opérations CRUD à l'aide d'un modèle de document.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
    /// <param name="tableName">The name of the table to load.</param>
    /// <returns>A DynamoDB table object.</returns>
```

```
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
```

```
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
```

```
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
```

```
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
```

```
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

Effectuez des opérations d'écriture par lots à l'aide d'un modèle de document.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
```

```
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
```

```
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

Analysez une table à l'aide d'un modèle de document.

```
   /// <summary>
   /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
   /// table for values.
   /// </summary>
   public class MidLevelScanOnly
   {
       public static async Task Main()
       {
           IAmazonDynamoDB client = new AmazonDynamoDBClient();

           Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");

           await FindProductsWithNegativePrice(productCatalogTable);
           await FindProductsWithNegativePriceWithConfig(productCatalogTable);
       }

       /// <summary>
       /// Retrieves any products that have a negative price in a DynamoDB
table.
       /// </summary>
       /// <param name="productCatalogTable">A DynamoDB table object.</param>
       public static async Task FindProductsWithNegativePrice(
         Table productCatalogTable)
       {
           // Assume there is a price error. So we scan to find items priced <
0.
           var scanFilter = new ScanFilter();
           scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

           Search search = productCatalogTable.Scan(scanFilter);

           do
           {
               var documentList = await search.GetNextSetAsync();
               Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");

               foreach (var document in documentList)
               {
                   PrintDocument(document);
```

```
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
```

```
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Interrogez et analysez une table à l'aide d'un modèle de document.

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
```

```
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
```

```
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing .............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));
```

```
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
       {
         "Message",
         "ReplyDateTime",
         "PostedBy",
       },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
       }

       /// <summary>
       /// Perform a query for replies made in the last 15 days using a DynamoDB
       /// QueryOperationConfig object.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadName">The bane of the thread that we are searching
       /// for replies.</param>
```

```
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            var config = new QueryOperationConfig()
            {
                Filter = filter,

                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
```

```
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser un modèle de persistance des objets de haut niveau pour DynamoDB à l'aide d'un SDK AWS

L'exemple de code suivant montre comment effectuer des opérations de création, de lecture, de mise à jour et de suppression (CRUD) et par lots à l'aide d'un modèle de persistance d'objets pour DynamoDB et d'un SDK.AWS

Pour plus d’informations, consultez la section Modèle de persistance des objets.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Effectuez des opérations CRUD à l'aide d'un modèle de persistance des objets de haut niveau.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };

        // Save the book to the ProductCatalog table.
        await context.SaveAsync(myBook);

        // Retrieve the book from the ProductCatalog table.
```

```
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

Effectuez des opérations d'écriture par lots à l'aide d'un modèle de persistance des objets de haut niveau.

```
    /// <summary>
    /// Performs high-level batch write operations to an Amazon DynamoDB table.
    /// This example was written using the AWS SDK for .NET version 3.7 and .NET
```

```
    /// Core 5.0.
    /// </summary>
    public class HighLevelBatchWriteItem
    {
        public static async Task SingleTableBatchWrite(IDynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                Isbn = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book3 in batch write",
            };

            Book book2 = new Book
            {
                Id = 903,
                InPublication = true,
                Isbn = "903-11-11-1111",
                PageCount = "200",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book4 in batch write",
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
```

```
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

Mappez des données arbitraires à une table à l'aide d'un modèle de persistance des objets de haut niveau.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
```

```
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
```

```
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

Interrogez et analysez une table à l'aide d'un modèle de persistance des objets de haut niveau.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }
```

```
        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn}
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The thread object containing the query
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
```

```
            IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
```

```
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
```

```
        }
    }
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser les opérations de compteur atomique dans DynamoDB avec un SDK AWS

Les exemples de code suivants montrent comment utiliser les opérations de compteurs atomiques dans DynamoDB.

- Incrémentation des compteurs de manière atomique à l'aide des opérations ADD et SET.
- Incrémentez en toute sécurité les compteurs qui n'existent peut-être pas.
- Mettez en œuvre un verrouillage optimiste pour les opérations de compteur.

Java

SDK pour Java 2.x

Démontrez les opérations de contre-attaque atomique en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Increments a counter using the ADD operation.
     *
```

```
     * <p>This method demonstrates how to use the ADD operation to atomically
     * increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithAdd(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #counterName :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter using the SET operation.
     *
     * <p>This method demonstrates how to use the SET operation with an
expression
     * to increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter safely, handling the case where the counter doesn't
exist yet.
     *
     * <p>This method demonstrates how to use if_not_exists to safely increment a
counter
     * that may not exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
```

```
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName =
if_not_exists(#counterName, :zero) + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build(),
                ":zero", AttributeValue.builder().n("0").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Decrements a counter safely, ensuring it doesn't go below zero.
     *
     * <p>This method demonstrates how to use a condition expression to safely
     * decrement a counter without going below zero.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param decrementValue The value to decrement by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation or if
the counter would go below zero
     */
    public static UpdateItemResponse decrementCounterSafely(
```

```
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int decrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName - :decrement")
            .conditionExpression("#counterName >= :decrement")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":decrement",

AttributeValue.builder().n(String.valueOf(decrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Compares the ADD and SET approaches for incrementing counters.
     *
     * <p>This method demonstrates the differences between using ADD and SET
     * for incrementing counters in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the comparison results
     */
    public static Map<String, Object> compareAddVsSet(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Reset counters to ensure a fair comparison
            UpdateItemRequest resetRequest = UpdateItemRequest.builder()
```

```
                .tableName(tableName)
                .key(key)
                .updateExpression("SET AddCounter = :zero, SetCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .build();

            dynamoDbClient.updateItem(resetRequest);

            // Increment with ADD
            long addStartTime = System.nanoTime();
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "AddCounter", 1);
            long addEndTime = System.nanoTime();
            long addDuration = addEndTime - addStartTime;

            // Increment with SET
            long setStartTime = System.nanoTime();
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "SetCounter", 1);
            long setEndTime = System.nanoTime();
            long setDuration = setEndTime - setStartTime;

            // Record results
            results.put("addResponse", addResponse);
            results.put("setResponse", setResponse);
            results.put("addDuration", addDuration);
            results.put("setDuration", setDuration);
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Gets the current value of a counter attribute.
     *
     * <p>Helper method to retrieve the current value of a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param counterName The name of the counter attribute
     * @return The counter value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Integer getCounterValue(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String counterName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(counterName)
            .build();

        // Perform the get operation
        GetItemResponse response = dynamoDbClient.getItem(request);

        // Return the counter value if it exists, otherwise null
        if (response.item() != null && response.item().containsKey(counterName))
{
            return Integer.parseInt(response.item().get(counterName).n());
        }

        return null;
    }
```

Exemple d'utilisation d'opérations de compteur atomique avec AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating atomic counter operations in
DynamoDB");

        try {
            // Example 1: Increment a counter using ADD
```

```
            System.out.println("\nExample 1: Incrementing a counter using ADD");
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "ViewCount", 1);

            System.out.println("Updated counter: " + addResponse.attributes());

            // Example 2: Increment a counter using SET
            System.out.println("\nExample 2: Incrementing a counter using SET");
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "LikeCount", 1);

            System.out.println("Updated counter: " + setResponse.attributes());

            // Example 3: Increment a counter safely
            System.out.println("\nExample 3: Incrementing a counter safely");
            UpdateItemResponse safeResponse =
incrementCounterSafely(dynamoDbClient, tableName, key, "ShareCount", 1);

            System.out.println("Updated counter: " + safeResponse.attributes());

            // Example 4: Decrement a counter safely
            System.out.println("\nExample 4: Decrementing a counter safely");
            try {
                UpdateItemResponse decrementResponse =
                    decrementCounterSafely(dynamoDbClient, tableName, key,
"InventoryCount", 1);

                System.out.println("Updated counter: " +
decrementResponse.attributes());
            } catch (DynamoDbException e) {
                if (e.getMessage().contains("ConditionalCheckFailed")) {
                    System.out.println("Cannot decrement counter below zero");
                } else {
                    throw e;
                }
            }

            // Example 5: Compare ADD vs SET
            System.out.println("\nExample 5: Comparing ADD vs SET");
            Map<String, Object> comparison = compareAddVsSet(dynamoDbClient,
tableName, key);

            if ((boolean) comparison.get("success")) {
```

```
                System.out.println("ADD duration: " +
comparison.get("addDuration") + " ns");
                System.out.println("SET duration: " +
comparison.get("setDuration") + " ns");
                System.out.println("ADD response: " +
comparison.get("addResponse"));
                System.out.println("SET response: " +
comparison.get("setResponse"));
            } else {
                System.out.println("Comparison failed: " +
comparison.get("error"));
            }

            // Explain atomic counter operations
            System.out.println("\nKey points about DynamoDB atomic counter
operations:");
            System.out.println("1. Both ADD and SET can be used for atomic
counters");
            System.out.println("2. ADD is more concise for simple increments");
            System.out.println("3. SET with an expression is more flexible for
complex operations");
            System.out.println("4. Use if_not_exists to handle the case where the
counter doesn't exist yet");
            System.out.println("5. Use condition expressions to prevent counters
from going below zero");
            System.out.println("6. Atomic operations are guaranteed to be
isolated from other writes");
            System.out.println("7. ADD can only be used with number and set data
types");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

Démontrez les opérations de contre-attaque atomique en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
```

```
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than
 using ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
```

```
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not
 exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} =
 if_not_exists(${counterName}, :default) + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
```

```
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement
 optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
```

```
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not
 found
 */
async function getCounterValue(
  config,
  tableName,
  key,
```

```
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Exemple d'utilisation d'opérations de compteur atomique avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
```

```
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key,
counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
```

```
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
 ${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex
 operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might
 not exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Python

Kit SDK for Python (Boto3)

Démontrez les opérations de contre-attaque atomique en utilisant AWS SDK pour Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Union


def increment_counter_with_add(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the ADD operation.

    This function demonstrates the atomic ADD operation, which is ideal for
    incrementing counters without the risk of race conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the ADD operation to atomically increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #counter :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_with_set(
```

```
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the SET operation with an expression.

    This function demonstrates using SET with an expression to increment a
 counter.
    While this works, it's generally recommended to use ADD for simple
 increments.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation with an expression to increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = #counter + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_safely(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    increment_value: int = 1,
    initial_value: int = 0,
```

```
) -> Dict[str, Any]:
    """
    Increment a counter attribute safely, handling the case where it might not
 exist.

    This function demonstrates a best practice for incrementing counters by using
    the if_not_exists function to handle the case where the counter doesn't exist
 yet.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.
        initial_value (int, optional): The initial value if the counter doesn't
 exist. Defaults to 0.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with if_not_exists to safely increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = if_not_exists(#counter, :initial)
 + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value, ":initial":
 initial_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def atomic_conditional_increment(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
```

```
    condition_attribute: str,
    condition_value: Any,
    increment_value: int = 1,
) -> Union[Dict[str, Any], None]:
    """
    Atomically increment a counter only if a condition is met.

    This function demonstrates combining atomic counter operations with
    conditional expressions for more complex update scenarios.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Optional[Dict[str, Any]]: The response from DynamoDB if successful, None
 if condition failed.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Use ADD with a condition expression
        response = table.update_item(
            Key=key,
            UpdateExpression="ADD #counter :increment",
            ConditionExpression="#condition = :value",
            ExpressionAttributeNames={"#counter": counter_name, "#condition":
 condition_attribute},
            ExpressionAttributeValues={":increment": increment_value, ":value":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
```

```
        else:
            # Other error occurred
            raise
```

Exemple d'utilisation d'opérations de compteur atomique avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the atomic counter operations functions."""
    # Example parameters
    table_name = "GameScores"
    key = {"UserId": "user123", "GameId": "game456"}
    counter_name = "Score"

    print("Example 1: Incrementing a counter with ADD operation")
    try:
        response = increment_counter_with_add(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=10
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with ADD: {e}")

    print("\nExample 2: Incrementing a counter with SET operation")
    try:
        response = increment_counter_with_set(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=5
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with SET: {e}")

    print("\nExample 3: Safely incrementing a counter that might not exist")
```

```
    try:
        new_key = {"UserId": "newuser789", "GameId": "game456"}
        response = increment_counter_safely(
            table_name=table_name,
            key=new_key,
            counter_name=counter_name,
            increment_value=15,
            initial_value=100,
        )
        print(
            f"Counter safely incremented. New value: {response.get('Attributes',
{}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error safely incrementing counter: {e}")

    print("\nExample 4: Conditional counter increment")
    try:
        # Fix for mypy: Handle the case where response might be None
        result = atomic_conditional_increment(
            table_name=table_name,
            key=key,
            counter_name="Achievements",
            condition_attribute="Level",
            condition_value=5,
            increment_value=1,
        )

        if result is not None:
            print(
                f"Conditional increment succeeded. New value:
{result.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
        if response:
            print(
                f"Conditional increment succeeded. New value:
{response.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
    except Exception as e:
        print(f"Error with conditional increment: {e}")
```

```
    print("\nComparison of ADD vs SET for counter operations:")
    print("1. ADD is specifically designed for atomic numeric increments and set
operations")
    print("2. SET with an expression can be used for more complex calculations")
    print("3. Both operations are atomic, preventing race conditions")
    print("4. ADD is more concise for simple increments")
    print("5. SET with if_not_exists() is recommended when the attribute might
not exist")
    print("6. For counters, ADD is generally preferred for clarity and
simplicity")
```

- Pour plus de détails sur l'API, consultez UpdateItemle AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser des opérations conditionnelles dans DynamoDB avec un SDK AWS

Les exemples de code suivants montrent comment utiliser des opérations conditionnelles dans DynamoDB.

- Implémentation d'écritures conditionnelles pour éviter le remplacement de données.
- Utilisez des expressions conditionnelles pour appliquer des règles de gestion.
- Gérez élégamment les échecs des vérifications conditionnelles.

Java

SDK pour Java 2.x

Démontrez les opérations conditionnelles en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Performs a conditional update on an item.
     *
     * <p>This method demonstrates how to use a condition expression to update an
 item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdate(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
```

```
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of(
                    "#condAttr", conditionAttribute,
                    "#updateAttr", updateAttribute))
                .expressionAttributeValues(Map.of(
                    ":condVal", conditionValue,
                    ":updateVal", updateValue))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and update was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, update was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional delete on an item.
     *
     * <p>This method demonstrates how to use a condition expression to delete an
item
     * only if a specific condition is met.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to delete
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalDelete(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the delete parameters
            DeleteItemRequest request = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of("#condAttr",
conditionAttribute))
                .expressionAttributeValues(Map.of(":condVal", conditionValue))
                .returnValues(ReturnValue.ALL_OLD)
                .build();

            // Perform the delete operation
            DeleteItemResponse response = dynamoDbClient.deleteItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and delete was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, delete was not
performed");
            result.put("error", "ConditionalCheckFailedException");
```

```
        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Demonstrates optimistic locking using a version attribute.
     *
     * <p>This method shows how to implement optimistic locking by using a
 version
     * attribute that is incremented with each update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param versionAttribute The name of the version attribute
     * @return Map containing the operation result
     */
    public static Map<String, Object> optimisticLockingExample(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
 AttributeValue> key, String versionAttribute) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Get the current version of the item
            GetItemRequest getRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .projectionExpression(versionAttribute)
                .build();

            GetItemResponse getResponse = dynamoDbClient.getItem(getRequest);

            // Check if the item exists
            if (getResponse.item() == null || !
getResponse.item().containsKey(versionAttribute)) {
                // Item doesn't exist or doesn't have a version attribute
                // Initialize with version 1
```

```
                UpdateItemRequest initRequest = UpdateItemRequest.builder()
                    .tableName(tableName)
                    .key(key)
                    .updateExpression("SET #verAttr = :newVer, #dataAttr
= :data")
                    .expressionAttributeNames(Map.of("#verAttr",
versionAttribute, "#dataAttr", "Data"))
                    .expressionAttributeValues(Map.of(
                        ":newVer", AttributeValue.builder().n("1").build(),
                        ":data", AttributeValue.builder().s("Initial
data").build()))
                    .returnValues(ReturnValue.UPDATED_NEW)
                    .build();

                UpdateItemResponse initResponse =
dynamoDbClient.updateItem(initRequest);

                result.put("operation", "initialize");
                result.put("success", true);
                result.put("attributes", initResponse.attributes());

                return result;
            }

            // Get the current version number
            int currentVersion =
                Integer.parseInt(getResponse.item().get(versionAttribute).n());
            int newVersion = currentVersion + 1;

            // Update the item with a condition on the version
            UpdateItemRequest updateRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #verAttr = :newVer, #dataAttr = :newData")
                .conditionExpression("#verAttr = :curVer")
                .expressionAttributeNames(Map.of("#verAttr", versionAttribute,
"#dataAttr", "Data"))
                .expressionAttributeValues(Map.of(
                    ":curVer",
                        AttributeValue.builder()
                            .n(String.valueOf(currentVersion))
                            .build(),
                    ":newVer",
```

```
AttributeValue.builder().n(String.valueOf(newVersion)).build(),
                    ":newData",
                        AttributeValue.builder()
                            .s("Updated data at version " + newVersion)
                            .build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse updateResponse =
dynamoDbClient.updateItem(updateRequest);

            // Record success result
            result.put("operation", "update");
            result.put("success", true);
            result.put("oldVersion", currentVersion);
            result.put("newVersion", newVersion);
            result.put("attributes", updateResponse.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to version mismatch
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Version mismatch, another process may have
updated the item");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional update with multiple conditions.
     *
     * <p>This method demonstrates how to use multiple conditions in a condition
expression.
     *
```

```
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditions Map of attribute names to values for conditions
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdateWithMultipleConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        Map<String, AttributeValue> conditions,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Build the condition expression and attribute names/values
            StringBuilder conditionExpression = new StringBuilder();
            Map<String, String> expressionAttributeNames = new HashMap<>();
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();

            // Add update attribute
            expressionAttributeNames.put("#updateAttr", updateAttribute);
            expressionAttributeValues.put(":updateVal", updateValue);

            // Add conditions
            int i = 0;
            for (Map.Entry<String, AttributeValue> condition :
conditions.entrySet()) {
                String attrName = condition.getKey();
                AttributeValue attrValue = condition.getValue();

                String nameKey = "#cond" + i;
                String valueKey = ":val" + i;

                expressionAttributeNames.put(nameKey, attrName);
                expressionAttributeValues.put(valueKey, attrValue);

                // Add AND between conditions (except for the first one)
                if (i > 0) {
```

```
                    conditionExpression.append(" AND ");
                }

                conditionExpression.append(nameKey).append(" =
").append(valueKey);
                i++;
            }

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression(conditionExpression.toString())
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "All conditions were met and update was
performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "One or more conditions were not met, update
was not performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }
```

Exemple d'utilisation d'opérations conditionnelles avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating conditional operations in DynamoDB");

       try {
           // Example 1: Conditional update
           System.out.println("\nExample 1: Conditional update");
           Map<String, Object> updateResult = conditionalUpdate(
               dynamoDbClient,
               tableName,
               key,
               "InStock",
               AttributeValue.builder().bool(true).build(),
               "Status",
               AttributeValue.builder().s("Available").build());

           System.out.println("Update result: " + updateResult.get("message"));
           if ((boolean) updateResult.get("success")) {
               System.out.println("Updated attributes: " +
updateResult.get("attributes"));
           }

           // Example 2: Conditional delete
           System.out.println("\nExample 2: Conditional delete");
           Map<String, Object> deleteResult = conditionalDelete(
               dynamoDbClient,
               tableName,
               key,
               "Status",
               AttributeValue.builder().s("Discontinued").build());

           System.out.println("Delete result: " + deleteResult.get("message"));
           if ((boolean) deleteResult.get("success")) {
               System.out.println("Deleted item: " +
deleteResult.get("attributes"));
```

```
            }

            // Example 3: Optimistic locking
            System.out.println("\nExample 3: Optimistic locking");
            Map<String, Object> lockingResult =
optimisticLockingExample(dynamoDbClient, tableName, key, "Version");

            System.out.println("Optimistic locking result:");
            System.out.println("  Operation: " + lockingResult.get("operation"));
            System.out.println("  Success: " + lockingResult.get("success"));
            if (lockingResult.get("operation").equals("update") && (boolean)
lockingResult.get("success")) {
                System.out.println("  Old version: " +
lockingResult.get("oldVersion"));
                System.out.println("  New version: " +
lockingResult.get("newVersion"));
            }
            System.out.println("  Attributes: " +
lockingResult.get("attributes"));

            // Example 4: Multiple conditions
            System.out.println("\nExample 4: Multiple conditions");
            Map<String, AttributeValue> conditions = new HashMap<>();
            conditions.put("Price",
AttributeValue.builder().n("199.99").build());
            conditions.put("Category",
AttributeValue.builder().s("Electronics").build());

            Map<String, Object> multiConditionResult =
conditionalUpdateWithMultipleConditions(
                dynamoDbClient,
                tableName,
                key,
                conditions,
                "OnSale",
                AttributeValue.builder().bool(true).build());

            System.out.println("Multiple conditions result: " +
multiConditionResult.get("message"));
            if ((boolean) multiConditionResult.get("success")) {
                System.out.println("Updated attributes: " +
multiConditionResult.get("attributes"));
            }
```

```
            // Explain conditional operations
            System.out.println("\nKey points about DynamoDB conditional
operations:");
            System.out.println("1. Conditional operations only succeed if the
condition is met");
            System.out.println("2. ConditionalCheckFailedException is thrown when
the condition fails");
            System.out.println("3. No changes are made to the item if the
condition fails");
            System.out.println("4. Conditions can be used with update, delete,
and put operations");
            System.out.println("5. Multiple conditions can be combined with AND
and OR");
            System.out.println("6. Optimistic locking can be implemented using a
version attribute");
            System.out.println(
                "7. Conditional operations consume the same amount of write
capacity whether they succeed or fail");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK for Java 2.x.
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK pour JavaScript (v3)

Démontrez les opérations conditionnelles en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
```

```
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
```

```
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
```

```
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );

  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }
```

```
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations conditionnelles avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use conditional operations.
```

```
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");
```

```
    // First, get the current item to check its version
    const currentItem = await getItem(config, tableName, { ProductId:
"P67890" });
    const currentVersion = currentItem ? (currentItem.version || 0) : 0;

    console.log(`Current version: ${currentVersion}`);

    // Then, update with optimistic locking
    const lockingResult = await updateWithOptimisticLocking(
      config,
      tableName,
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }
```

```
    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is
 met");
    console.log("2. ConditionalCheckFailedException indicates the condition
 wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript.
  - DeleteItem
  - PutItem
  - UpdateItem

Python

Kit SDK for Python (Boto3)

Démontrez les opérations conditionnelles en utilisant AWS SDK pour Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Optional, Tuple, Union
```

```
def conditional_update(
    table_name: str,
    key: Dict[str, Any],
    condition_attribute: str,
    condition_value: Any,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item only if a condition is met.

    This function demonstrates how to perform a conditional update operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression="SET #update_attr = :update_val",
            ConditionExpression="#cond_attr = :cond_val",
            ExpressionAttributeNames={
                "#update_attr": update_attribute,
                "#cond_attr": condition_attribute,
            },
            ExpressionAttributeValues={":update_val": update_value, ":cond_val":
 condition_value},
            ReturnValues="UPDATED_NEW",
```

```
        )
        # Update succeeded, condition was met
        return True, response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False, None
        else:
            # Other error occurred
            raise


def conditional_delete(
    table_name: str, key: Dict[str, Any], condition_attribute: str,
 condition_value: Any
) -> bool:
    """
    Delete an item only if a condition is met.

    This function demonstrates how to perform a conditional delete operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to delete.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.

    Returns:
        bool: True if the delete succeeded (condition was met), False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional delete
        table.delete_item(
            Key=key,
            ConditionExpression="#attr = :val",
            ExpressionAttributeNames={"#attr": condition_attribute},
            ExpressionAttributeValues={":val": condition_value},
        )
        # Delete succeeded, condition was met
```

```
        return True
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False
        else:
            # Other error occurred
            raise


def optimistic_locking_update(
    table_name: str,
    key: Dict[str, Any],
    version_attribute: str,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item using optimistic locking with a version attribute.

    This function demonstrates how to implement optimistic locking using
    a version attribute that is incremented with each update.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        version_attribute (str): The name of the version attribute.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # First, get the current version
    try:
        response = table.get_item(
            Key=key,
            ProjectionExpression=f"#{version_attribute}",
```

```
            ExpressionAttributeNames={f"#{version_attribute}":
 version_attribute},
        )

        item = response.get("Item", {})
        current_version = item.get(version_attribute, 0)

        # Now, try to update with a condition on the version
        try:
            update_response = table.update_item(
                Key=key,
                UpdateExpression=f"SET #{update_attribute} = :update_val,
 #{version_attribute} = :new_version",
                ConditionExpression=f"#{version_attribute} = :current_version",
                ExpressionAttributeNames={
                    f"#{update_attribute}": update_attribute,
                    f"#{version_attribute}": version_attribute,
                },
                ExpressionAttributeValues={
                    ":update_val": update_value,
                    ":current_version": current_version,
                    ":new_version": current_version + 1,
                },
                ReturnValues="UPDATED_NEW",
            )
            # Update succeeded
            return True, update_response
        except ClientError as e:
            if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
                # Version has changed, optimistic locking failed
                return False, None
            else:
                # Other error occurred
                raise
    except ClientError:
        # Error getting the item
        raise


def conditional_check_and_update(
    table_name: str,
    key: Dict[str, Any],
    check_attribute: str,
    check_value: Any,
```

```
    update_attribute: str,
    update_value: Any,
    create_if_not_exists: bool = False,
) -> Union[Dict[str, Any], None]:
    """
    Check if an attribute has a specific value and update another attribute if it
 does.

    This function demonstrates a more complex conditional update that can also
    create the item if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        check_attribute (str): The attribute to check in the condition.
        check_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.
        create_if_not_exists (bool, optional): Whether to create the item if it
 doesn't exist.

    Returns:
        Union[Dict[str, Any], None]: The response from DynamoDB if successful,
 None otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        if create_if_not_exists:
            # Use attribute_not_exists to create the item if it doesn't exist
            condition_expression = "attribute_not_exists(#pk) OR #check_attr
 = :check_val"
            update_expression = "SET #update_attr = :update_val, #check_attr =
 if_not_exists(#check_attr, :check_val)"

            # Get the partition key name from the key dictionary
            pk_name = next(iter(key))

            expression_attribute_names = {
                "#pk": pk_name,
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
```

```
            }
        else:
            # Only update if the check attribute has the expected value
            condition_expression = "#check_attr = :check_val"
            update_expression = "SET #update_attr = :update_val"

            expression_attribute_names = {
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }

        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues={":check_val": check_value, ":update_val":
 update_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Exemple d'utilisation d'opérations conditionnelles avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use the conditional operations functions."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Conditional Update")
    try:
```

```
        # Update the price only if the current stock is greater than 10
        success, response = conditional_update(
            table_name=table_name,
            key=key,
            condition_attribute="Stock",
            condition_value=10,
            update_attribute="Price",
            update_value=99.99,
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Update succeeded! New values: {attributes}")
        else:
            print("Update failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional update: {e}")

    print("\nExample 2: Conditional Delete")
    try:
        # Delete the product only if it's discontinued
        success = conditional_delete(
            table_name=table_name,
            key=key,
            condition_attribute="Status",
            condition_value="Discontinued",
        )

        if success:
            print("Delete succeeded! The item was deleted.")
        else:
            print("Delete failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional delete: {e}")

    print("\nExample 3: Optimistic Locking")
    try:
        # Update with optimistic locking using a version attribute
        success, response = optimistic_locking_update(
            table_name=table_name,
            key=key,
            version_attribute="Version",
```

```
            update_attribute="Description",
            update_value="Updated product description",
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Optimistic locking update succeeded! New values:
{attributes}")
        else:
            print("Optimistic locking update failed because the version has
changed.")
    except Exception as e:
        print(f"Error during optimistic locking update: {e}")

    print("\nExample 4: Conditional Check and Update")
    try:
        # Update the featured status if the product is in stock
        response = conditional_check_and_update(
            table_name=table_name,
            key=key,
            check_attribute="InStock",
            check_value=True,
            update_attribute="Featured",
            update_value=True,
            create_if_not_exists=True,
        )

        if response:
            print(
                f"Conditional check and update succeeded! New values:
{response.get('Attributes', {})}"
            )
        else:
            print("Conditional check and update failed because the condition was
not met.")
    except Exception as e:
        print(f"Error during conditional check and update: {e}")

    print("\nUnderstanding Conditional Operations in DynamoDB:")
    print("1. Conditional operations help maintain data integrity")
    print("2. They prevent race conditions in concurrent environments")
    print("3. Failed conditions result in ConditionalCheckFailedException")
```

```
    print("4. No DynamoDB capacity is consumed when conditions fail")
    print("5. Optimistic locking is a common pattern using version attributes")
    print("6. Conditions can be combined with logical operators (AND, OR, NOT)")
    print("7. Conditions can use comparison operators (=, <>, <, <=, >, >=)")
    print(
        "8. attribute_exists() and attribute_not_exists() are useful for checking
attribute presence"
    )
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser les noms d'attributs d'expression dans DynamoDB avec un SDK AWS

Les exemples de code suivants montrent comment utiliser les noms d'attributs d'expression dans DynamoDB.

- Utilisation de mots réservés dans les expressions DynamoDB.
- Utilisez des espaces réservés pour les noms d'attributs d'expression.
- Gérez des caractères spéciaux dans les noms d'attributs.

Java

SDK pour Java 2.x

Démontrez les noms d'attributs d'expression en utilisant AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates an attribute that is a reserved word in DynamoDB.
     *
     * <p>This method demonstrates how to use expression attribute names to
 update
     * attributes that are reserved words in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param reservedWordAttribute The reserved word attribute to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", reservedWordAttribute))
```

```
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates an attribute that contains special characters.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * attributes that contain special characters.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param specialCharAttribute The attribute with special characters to
update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateSpecialCharacterAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String specialCharAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", specialCharAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }
```

```
   /**
    * Queries items using an attribute that is a reserved word.
    *
    * <p>This method demonstrates how to use expression attribute names in a
query
    * when the attribute is a reserved word.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param partitionKeyName The name of the partition key attribute
    * @param partitionKeyValue The value of the partition key
    * @param reservedWordAttribute The reserved word attribute to filter on
    * @param value The value to compare against
    * @return The response from DynamoDB
    * @throws DynamoDbException if an error occurs during the operation
    */
   public static QueryResponse queryWithReservedWordAttribute(
       DynamoDbClient dynamoDbClient,
       String tableName,
       String partitionKeyName,
       AttributeValue partitionKeyValue,
       String reservedWordAttribute,
       AttributeValue value) {

       // Define the query parameters using expression attribute names
       Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put("#pkName", partitionKeyName);
       expressionAttributeNames.put("#attr", reservedWordAttribute);

       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
       expressionAttributeValues.put(":pkValue", partitionKeyValue);
       expressionAttributeValues.put(":value", value);

       QueryRequest request = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression("#pkName = :pkValue")
           .filterExpression("#attr = :value")
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .build();

       // Perform the query operation
       return dynamoDbClient.query(request);
```

```
    }

    /**
     * Updates a nested attribute with a path that contains reserved words.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * nested attributes where the path contains reserved words.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributePath The path to the nested attribute as an array
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateNestedReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> attributePath,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < attributePath.size(); i++) {
            expressionAttributeNames.put("#attr" + i, attributePath.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < attributePath.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
```

```
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Scans a table with multiple attribute name placeholders.
     *
     * <p>This method demonstrates how to use multiple expression attribute names
     * in a complex filter expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param filters Object mapping attribute names to filter values
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithMultipleAttributeNames(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> filters) {

        // Create expression attribute names and values
        Map<String, String> expressionAttributeNames = new HashMap<>();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        StringBuilder filterExpression = new StringBuilder();

        // Build the filter expression
        int index = 0;
        for (Map.Entry<String, AttributeValue> entry : filters.entrySet()) {
            String attrName = entry.getKey();
            AttributeValue attrValue = entry.getValue();

            String nameKey = "#attr" + index;
            String valueKey = ":val" + index;

            expressionAttributeNames.put(nameKey, attrName);
            expressionAttributeValues.put(valueKey, attrValue);
```

```
            // Add AND between conditions (except for the first one)
            if (index > 0) {
                filterExpression.append(" AND ");
            }

            filterExpression.append(nameKey).append(" = ").append(valueKey);
            index++;
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression(filterExpression.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }
```

Exemple d'utilisation de noms d'attributs d'expression avec AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating expression attribute names in
DynamoDB");

        try {
            // Example 1: Update an attribute that is a reserved word
            System.out.println("\nExample 1: Updating an attribute that is a
reserved word");
            UpdateItemResponse response1 = updateReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Size", // "SIZE" is a reserved word in DynamoDB
                AttributeValue.builder().s("Large").build());
```

```
            System.out.println("Updated attribute: " + response1.attributes());

            // Example 2: Update an attribute with special characters
            System.out.println("\nExample 2: Updating an attribute with special
characters");
            UpdateItemResponse response2 = updateSpecialCharacterAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Product-Type", // Contains a hyphen, which is a special
character
                AttributeValue.builder().s("Electronics").build());

            System.out.println("Updated attribute: " + response2.attributes());

            // Example 3: Query with a reserved word attribute
            System.out.println("\nExample 3: Querying with a reserved word
attribute");
            QueryResponse response3 = queryWithReservedWordAttribute(
                dynamoDbClient,
                tableName,
                "Category",
                AttributeValue.builder().s("Electronics").build(),
                "Count", // "COUNT" is a reserved word in DynamoDB
                AttributeValue.builder().n("10").build());

            System.out.println("Found " + response3.count() + " items");

            // Example 4: Update a nested attribute with reserved words in the
path
            System.out.println("\nExample 4: Updating a nested attribute with
reserved words in the path");
            UpdateItemResponse response4 = updateNestedReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                Arrays.asList("Dimensions", "Size", "Height"), // "SIZE" is a
reserved word
                AttributeValue.builder().n("30").build());

            System.out.println("Updated nested attribute: " +
response4.attributes());
```

```
            // Example 5: Scan with multiple attribute name placeholders
            System.out.println("\nExample 5: Scanning with multiple attribute
 name placeholders");
            Map<String, AttributeValue> filters = new HashMap<>();
            filters.put("Size", AttributeValue.builder().s("Large").build());
            filters.put("Count", AttributeValue.builder().n("10").build());
            filters.put(
                "Product-Type",
 AttributeValue.builder().s("Electronics").build());

            ScanResponse response5 =
 scanWithMultipleAttributeNames(dynamoDbClient, tableName, filters);

            System.out.println("Found " + response5.count() + " items");

            // Show some common reserved words
            System.out.println("\nSome common DynamoDB reserved words:");
            List<String> commonReservedWords = getDynamoDBReservedWords();
            System.out.println(String.join(", ", commonReservedWords));

            // Explain expression attribute names
            System.out.println("\nKey points about expression attribute names:");
            System.out.println("1. Use expression attribute names (#name) for
 reserved words");
            System.out.println("2. Use expression attribute names for attributes
 with special characters");
            System.out.println(
                "3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
            System.out.println("4. Expression attribute names are required for
 nested attributes with reserved words");
            System.out.println("5. You can use multiple expression attribute
 names in a single expression");
            System.out.println("6. Expression attribute names are case-
sensitive");
            System.out.println("7. Expression attribute names are only used in
 expressions, not in the actual data");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x.
  - Interrogation
  - UpdateItem

JavaScript

SDK pour JavaScript (v3)

Démontrez les noms d'attributs d'expression en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters
 to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter
 on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
```

```
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
```

```
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
```

```
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation de noms d'attributs d'expression avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
```

```
const tableName = "Products";
const key = { ProductId: "P12345" };

console.log("Demonstrating expression attribute names in DynamoDB");

try {
  // Example 1: Update an attribute that is a reserved word
  console.log("\nExample 1: Updating an attribute that is a reserved word");
  const response1 = await updateReservedWordAttribute(
    config,
    tableName,
    key,
    "Size", // "SIZE" is a reserved word in DynamoDB
    "Large"
  );

  console.log("Updated attribute:", response1.Attributes);

  // Example 2: Update an attribute with special characters
  console.log("\nExample 2: Updating an attribute with special characters");
  const response2 = await updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    "Product-Type", // Contains a hyphen, which is a special character
    "Electronics"
  );

  console.log("Updated attribute:", response2.Attributes);

  // Example 3: Query with a reserved word attribute
  console.log("\nExample 3: Querying with a reserved word attribute");
  const response3 = await queryWithReservedWordAttribute(
    config,
    tableName,
    "Category",
    "Electronics",
    "Count", // "COUNT" is a reserved word in DynamoDB
    10
  );

  console.log(`Found ${response3.Items.length} items`);

  // Example 4: Update a nested attribute with reserved words in the path
```

```
   console.log("\nExample 4: Updating a nested attribute with reserved words in
the path");
   const response4 = await updateNestedReservedWordAttribute(
     config,
     tableName,
     key,
     ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
     30
   );

   console.log("Updated nested attribute:", response4.Attributes);

   // Example 5: Scan with multiple attribute name placeholders
   console.log("\nExample 5: Scanning with multiple attribute name
placeholders");
   const response5 = await scanWithMultipleAttributeNames(
     config,
     tableName,
     {
       "Size": "Large",
       "Count": 10,
       "Product-Type": "Electronics"
     }
   );

   console.log(`Found ${response5.Items.length} items`);

   // Get the final state of the item
   console.log("\nFinal state of the item:");
   const item = await getItem(config, tableName, key);
   console.log(JSON.stringify(item, null, 2));

   // Show some common reserved words
   console.log("\nSome common DynamoDB reserved words:");
   const commonReservedWords = [
     "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
     "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
     "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
     "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
     "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN",
"NAME",
     "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
     "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
     "VIEW", "WHERE"
```

```
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not
 in the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript.
  - Interrogation
  - UpdateItem

Python

Kit SDK for Python (Boto3)

Démontrez les noms d'attributs d'expression en utilisant AWS SDK pour Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List


def use_reserved_word_attribute(
    table_name: str, key: Dict[str, Any], reserved_word: str, value: Any
```

```
) -> Dict[str, Any]:
    """
    Update an attribute whose name is a DynamoDB reserved word.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names that are DynamoDB reserved words.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        reserved_word (str): The reserved word to use as an attribute name.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle the reserved word
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #reserved_attr = :value",
        ExpressionAttributeNames={"#reserved_attr": reserved_word},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def use_special_character_attribute(
    table_name: str, key: Dict[str, Any], attribute_with_special_chars: str,
 value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name contains special characters.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names containing special characters like spaces, dots,
 or hyphens.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attribute_with_special_chars (str): The attribute name with special
 characters.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle special characters
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #special_attr = :value",
        ExpressionAttributeNames={"#special_attr": attribute_with_special_chars},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def query_with_attribute_names(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    filter_attribute_name: str,
    filter_value: Any,
) -> Dict[str, Any]:
    """
    Query a table using expression attribute names for both key and filter
 attributes.

    This function demonstrates how to use expression attribute names in a query
 operation
    for both the key condition expression and filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute_name (str): The name of the attribute to filter on.
        filter_value (Any): The value to compare against in the filter.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names for both key condition and filter
    response = table.query(
        KeyConditionExpression="#pk = :pk_val",
        FilterExpression="#filter_attr = :filter_val",
        ExpressionAttributeNames={"#pk": partition_key_name, "#filter_attr":
 filter_attribute_name},
        ExpressionAttributeValues={":pk_val": partition_key_value, ":filter_val":
 filter_value},
    )

    return response


def update_nested_attribute_with_dots(
    table_name: str, key: Dict[str, Any], path_with_dots: str, value: Any
) -> Dict[str, Any]:
    """
    Update a nested attribute using a path with dot notation.

    This function demonstrates how to use expression attribute names to work with
    nested attributes specified using dot notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path_with_dots (str): The path to the nested attribute using dot notation
 (e.g., "a.b.c").
        value (Any): The value to set for the nested attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path_with_dots.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def demonstrate_attribute_name_requirements(table_name: str, key: Dict[str, Any])
 -> Dict[str, Any]:
    """
    Demonstrate the requirements and allowed characters for attribute names.

    This function shows examples of valid and invalid attribute names and how to
```

```
    handle them using expression attribute names.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Examples of attribute names with different characteristics
    examples = {
        "valid_standard": "NormalAttribute",  # Standard attribute name (no
placeholder needed)
        "valid_with_underscore": "Normal_Attribute",  # Underscore is allowed
        "valid_with_number": "Attribute123",  # Numbers are allowed
        "reserved_word": "Timestamp",  # Reserved word (requires placeholder)
        "starts_with_number": "123Attribute",  # Starts with number (valid but
may need placeholder in some contexts)
        "with_space": "Attribute Name",  # Contains space (requires placeholder)
        "with_dot": "Attribute.Name",  # Contains dot (requires placeholder)
        "with_hyphen": "Attribute-Name",  # Contains hyphen (requires
placeholder)
        "with_special_chars": "Attribute#$%",  # Contains special characters
(requires placeholder)
    }

    results = {}

    # Try to update each attribute type
    for example_type, attr_name in examples.items():
        try:
            # For attributes that don't need placeholders, try direct reference
            if example_type in ["valid_standard", "valid_with_underscore",
"valid_with_number"]:
                try:
                    # Try without expression attribute names first
                    response = table.update_item(
                        Key=key,
                        UpdateExpression=f"SET {attr_name} = :value",
```

```
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": False,
                        "response": response,
                    }
                except ClientError:
                    # If direct reference fails, try with placeholder
                    response = table.update_item(
                        Key=key,
                        UpdateExpression="SET #attr = :value",
                        ExpressionAttributeNames={"#attr": attr_name},
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": True,
                        "response": response,
                    }
            else:
                # For attributes that definitely need placeholders
                response = table.update_item(
                    Key=key,
                    UpdateExpression="SET #attr = :value",
                    ExpressionAttributeNames={"#attr": attr_name},
                    ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                    ReturnValues="UPDATED_NEW",
                )
                results[example_type] = {
                    "attribute_name": attr_name,
                    "success": True,
                    "needed_placeholder": True,
                    "response": response,
                }
        except ClientError as e:
```

```
            results[example_type] = {"attribute_name": attr_name, "success":
 False, "error": str(e)}

    return results
```

Exemple d'utilisation de noms d'attributs d'expression avec AWS SDK pour Python (Boto3).

```
def example_usage():
    """Example of how to use expression attribute names in DynamoDB."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Using a reserved word as an attribute name")
    try:
        response = use_reserved_word_attribute(
            table_name=table_name, key=key, reserved_word="Timestamp",
 value="2025-05-14T12:00:00Z"
        )
        print(f"Reserved word attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating reserved word attribute: {e}")

    print("\nExample 2: Using an attribute name with special characters")
    try:
        response = use_special_character_attribute(
            table_name=table_name,
            key=key,
            attribute_with_special_chars="Product Info",
            value="Special product information",
        )
        print(f"Special character attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating special character attribute: {e}")

    print("\nExample 3: Querying with expression attribute names")
    try:
        response = query_with_attribute_names(
```

```
            table_name=table_name,
            partition_key_name="Category",
            partition_key_value="Electronics",
            filter_attribute_name="Price",
            filter_value=500,
        )
        print(
            f"Query with expression attribute names returned
{len(response.get('Items', []))} items"
        )
    except Exception as e:
        print(f"Error querying with expression attribute names: {e}")

    print("\nExample 4: Updating a nested attribute with dot notation")
    try:
        response = update_nested_attribute_with_dots(
            table_name=table_name,
            key=key,
            path_with_dots="Product.Details.Specifications",
            value={"Weight": "2.5 kg", "Dimensions": "30x20x10 cm"},
        )
        print(f"Nested attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating nested attribute: {e}")

    print("\nExample 5: Demonstrating attribute name requirements")
    try:
        results = demonstrate_attribute_name_requirements(table_name=table_name,
key=key)

        print("Attribute Name Requirements Results:")
        for example_type, result in results.items():
            if result.get("success", False):
                needed_placeholder = result.get("needed_placeholder", True)
                print(
                    f"  - {example_type}: '{result['attribute_name']}' -
{'Requires' if needed_placeholder else 'Does not require'} placeholder"
                )
            else:
                print(
                    f"  - {example_type}: '{result['attribute_name']}' - Failed:
{result.get('error', 'Unknown error')}"
                )
```

```
    except Exception as e:
        print(f"Error demonstrating attribute name requirements: {e}")

    print("\nCommon DynamoDB Reserved Words (sample):")
    reserved_words = get_common_reserved_words()
    print(", ".join(reserved_words[:20]) + "... (and many more)")

    print("\nWhen to Use Expression Attribute Names:")
    print("1. When the attribute name is a DynamoDB reserved word")
    print("2. When the attribute name contains special characters (spaces, dots,
hyphens)")
    print("3. When the attribute name begins with a number")
    print("4. When working with nested attributes using dot notation")
    print("5. When you need to reference the same attribute multiple times in an
expression")

    print("\nExpression Attribute Name Requirements:")
    print("1. Must begin with a pound sign (#)")
    print("2. After the pound sign, must contain at least one character")
    print("3. Can contain alphanumeric characters and underscore (_)")
    print("4. Are case-sensitive")
    print("5. Must be unique within a single expression")

    print("\nAttribute Name Requirements in DynamoDB:")
    print("1. Can begin with a-z, A-Z, or 0-9")
    print("2. Can contain a-z, A-Z, 0-9, underscore (_), dash (-), and dot (.)")
    print("3. Are case-sensitive")
    print("4. No length restrictions, but practical limits apply")
    print("5. Cannot be a DynamoDB reserved word if used directly in
expressions")
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour Python (Boto3).
  - Interrogation
  - UpdateItem

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisent des événements planifiés pour invoquer une fonction Lambda

Les exemples de code suivants montrent comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

Java

SDK pour Java 2.x

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK pour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction

Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

Kit SDK for Python (Boto3)

Cet exemple montre comment enregistrer une AWS Lambda fonction en tant que cible d'un EventBridge événement Amazon planifié. Le gestionnaire Lambda écrit un message convivial et les données complètes de l'événement dans Amazon CloudWatch Logs pour une récupération ultérieure.

- Déploie une fonction Lambda.
- Crée un événement EventBridge planifié et fait de la fonction Lambda la cible.
- Accorde l'autorisation de laisser EventBridge invoquer la fonction Lambda.
- Imprime les dernières données des CloudWatch journaux pour afficher le résultat des appels planifiés.
- Nettoie toutes les ressources créées lors de la démonstration.

Il est préférable de visionner cet exemple sur GitHub. Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation des index secondaires locaux DynamoDB à l'aide de la version v2 AWS Command Line Interface

L'exemple de code suivant montre comment créer et interroger des tables avec des index secondaires locaux.

- Créez une table avec un index secondaire local (LSI).
- Créez un tableau à plusieurs LSIs avec différents types de projection.
- Interrogez les données en utilisant LSIs.

Bash

AWS CLI avec le script Bash

Créer une table avec un index secondaire local.

```
# Create a table with a Local Secondary Index
aws dynamodb create-table \
    --table-name CustomerOrders \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=OrderID,AttributeType=S \
        AttributeName=OrderDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=OrderID,KeyType=RANGE \
    --local-secondary-indexes \
```

```
        "IndexName=OrderDateIndex,\
        KeySchema=[{AttributeName=CustomerID,KeyType=HASH},
{AttributeName=OrderDate,KeyType=RANGE}],\
        Projection={ProjectionType=ALL}" \
    --billing-mode PAY_PER_REQUEST
```

Créez une table avec plusieurs LSIs.

```
# Create a table with multiple Local Secondary Indexes
aws dynamodb create-table \
    --table-name CustomerDetails \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=Name,AttributeType=S \
        AttributeName=Email,AttributeType=S \
        AttributeName=RegistrationDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=Name,KeyType=RANGE \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"EmailIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Email\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Address\",\"Phone\"]}
            },
            {
                \"IndexName\": \"RegistrationIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"RegistrationDate\",\"KeyType\":\"RANGE
\"}
                ],
                \"Projection\": {\"ProjectionType\":\"KEYS_ONLY\"}
            }
        ]" \
    --billing-mode PAY_PER_REQUEST
```

Interrogez les données en utilisant LSIs.

```
# Query the OrderDateIndex LSI
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId AND OrderDate BETWEEN :date1
 AND :date2" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":date1": {"S": "2023-01-01"},
        ":date2": {"S": "2023-02-01"}
    }'

# Query with a filter expression
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId" \
    --filter-expression "Amount > :amount" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":amount": {"N": "150"}
    }'
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - Interrogation

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisation de DynamoDB Streams et utilisation de la version 2 Time-to-Live AWS Command Line Interface

L'exemple de code suivant montre comment gérer les flux et les fonctionnalités de DynamoDB. Time-to-Live

- Créez une table avec Streams activé.
- Décrivez Streams.
- Créez une fonction Lambda pour traiter Streams.
- Activation du TTL sur une table.
- Ajoutez des éléments dotés d'attributs TTL.
- Décrivez les paramètres TTL.

Bash

AWS CLI avec le script Bash

Créez une table avec Streams activé.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Décrivez Streams.

```
# Get information about the stream
aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
```

```
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Créez une fonction Lambda pour Streams.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole

# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

Activation du TTL sur une table.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo

# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

Ajoutez des éléments dotés d'attributs TTL.

```
# Calculate expiration time (current time + 1 day in seconds)
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

Décrivez les paramètres TTL.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utilisez les tables globales DynamoDB et la réplication multirégionale avec une cohérence éventuelle (MREC) à l'aide de la version v2 AWS Command Line Interface

L'exemple de code suivant montre comment gérer les tables globales DynamoDB avec la cohérence à terme de la réplication multirégionale (MREC).

- Créez une table avec la réplication multirégionale (MREC).
- Placez et récupérez des objets de tables de réplica.
- Supprimez les répliques one-by-one.
- Nettoyez en supprimant la table.

Bash

AWS CLI avec le script Bash

Créez une table avec la réplication multirégionale.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
```

```
  ]
}' \
--region us-east-2
```

Décrivez la table multirégionale.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Placez les objets dans une table de réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Récupérez des objets de tables de réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Supprimez les réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Nettoyez en supprimant la table.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- UpdateTable

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser le balisage des ressources DynamoDB à l'aide de la version v2 AWS Command Line Interface

L'exemple de code suivant montre comment gérer des balises pour des ressources DynamoDB.

- Créez une table avec des balises.
- Répertoriez les balises d'une ressource.
- Ajoutez des balises à une ressource.
- Supprimer des balises d'une ressource.
- Filtrez les tables par balises.

Bash

AWS CLI avec le script Bash

Créez une table avec des balises.

```
# Create a table with tags
aws dynamodb create-table \
    --table-name TaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
```

```
        Key=Environment,Value=Production \
        Key=Project,Value=Analytics \
        Key=Owner,Value=DataTeam
```

Répertoriez les balises d'une ressource.

```
# Get the table ARN
TABLE_ARN=$(aws dynamodb describe-table \
    --table-name TaggedTable \
    --query "Table.TableArn" \
    --output text)

# List tags for the table
aws dynamodb list-tags-of-resource \
    --resource-arn $TABLE_ARN
```

Ajoutez des balises à une ressource.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Supprimer des balises d'une ressource.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Filtrez les tables par balises.

```
# Create another table with different tags
aws dynamodb create-table \
    --table-name AnotherTaggedTable \
```

```
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Development \
        Key=Project,Value=Testing

# Wait for table to become active
aws dynamodb wait table-exists --table-name AnotherTaggedTable

# List all tables
echo "All tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nFiltering tables by Environment=Production tag:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# Find tables with specific tag
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Utiliser le chiffrement des tables DynamoDB à l'aide de la version v2 AWS Command Line Interface

L'exemple de code suivant montre comment gérer les options de chiffrement pour les tables DynamoDB.

- Créez une table avec le chiffrement par défaut.
- Créez une table avec une clé CMK gérée par le client.
- Mettez à jour les paramètres de chiffrement des tables.
- Décrivez le chiffrement des tables.

Bash

AWS CLI avec le script Bash

Créez une table avec le chiffrement par défaut.

```
# Create a table with default encryption (AWS owned key)
aws dynamodb create-table \
    --table-name CustomerData \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS
```

Créez une table avec une clé CMK gérée par le client.

```
# Step 1: Create a customer managed key in KMS
aws kms create-key \
    --description "Key for DynamoDB table encryption" \
    --key-usage ENCRYPT_DECRYPT \
    --customer-master-key-spec SYMMETRIC_DEFAULT
```

```
# Store the key ID for later use
KEY_ID=$(aws kms list-keys --query "Keys[?contains(KeyArn, 'Key for
 DynamoDB')].KeyId" --output text)

# Step 2: Create a table with the customer managed key
aws dynamodb create-table \
    --table-name SensitiveData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Mettez à jour le chiffrement de la table.

```
# Update a table to use a different KMS key
aws dynamodb update-table \
    --table-name CustomerData \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Décrivez le chiffrement des tables.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des commandes de l'AWS CLI.
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Exemples sans serveur pour DynamoDB

Les exemples de code suivants montrent comment utiliser DynamoDB avec.AWS SDKs

Exemples

- Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB
- Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB

Les exemples de code suivants illustrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux DynamoDB. La fonction récupère les données utiles DynamoDB et journalise le contenu de l'enregistrement.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consommation d'un événement DynamoDB avec Lambda en utilisant. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data
```

```
            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
```

```
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

Les exemples de code suivants illustrent comment implémenter une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux DynamoDB. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
```

```
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

Kit SDK pour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""
```

```
 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;
```

```
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
```

```
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

Kit SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

Kit SDK for Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
```

```
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

AWS contributions de la communauté pour DynamoDB

AWS les contributions communautaires sont des exemples qui ont été créés et sont maintenus par plusieurs équipes AWS. Pour faire un commentaire, utilisez le mécanisme fourni dans les référentiels liés.

Exemples

- Création et test d’une application sans serveur

Création et test d’une application sans serveur

Les exemples de code suivants montrent comment créer et tester une application sans serveur à l’aide d’API Gateway avec Lambda et DynamoDB.

.NET

SDK pour .NET

Illustre comment créer et tester une application sans serveur composée d’une API Gateway avec Lambda et DynamoDB à l’aide du kit SDK .NET.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Go

Kit SDK pour Go V2

Illustre comment créer et tester une application sans serveur composée d’une API Gateway avec Lambda et DynamoDB à l’aide du kit SDK Go.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Java

SDK pour Java 2.x

Illustre comment créer et tester une application sans serveur composée d'une API Gateway avec Lambda et DynamoDB à l'aide du kit SDK Java.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Rust

SDK pour Rust

Illustre comment créer et tester une application sans serveur composée d'une API Gateway avec Lambda et DynamoDB à l'aide du kit SDK Rust.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation de DynamoDB avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes du kit SDK.

Sécurité et conformité dans Amazon DynamoDB

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- Sécurité du cloud :AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud.AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. L'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de conformitéAWS. Pour en savoir plus sur les programmes de conformité qui s'appliquent à DynamoDB, consultez Services AWS concernés par le programme de conformité.
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre organisation, et la législation et la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de DynamoDB. Les rubriques suivantes vous montrent comment configurer DynamoDB pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui peuvent vous aider à surveiller et à sécuriser vos ressources DynamoDB.

Rubriques

- AWS politiques gérées pour Amazon DynamoDB
- Utilisation de politiques basées sur des ressources pour DynamoDB
- Utilisation du contrôle d'accès par attributs avec DynamoDB
- Protection des données dans DynamoDB
- Gestion des identités et des accès AWS(IAM) et DynamoDB
- Validation de la conformité par l'industrie pour DynamoDB
- Résilience et reprise après sinistre dans Amazon DynamoDB
- Sécurité de l'infrastructure dans Amazon DynamoDB

- AWS PrivateLink pour DynamoDB
- Configuration et analyse des vulnérabilités dans Amazon DynamoDB
- Bonnes pratiques de sécurité pour Amazon DynamoDB

AWS politiques gérées pour Amazon DynamoDB

DynamoDB AWS utilise des politiques gérées pour définir un ensemble d'autorisations dont le service a besoin pour effectuer des actions spécifiques. DynamoDB gère et met à jour AWS ses politiques gérées. Vous ne pouvez pas modifier les autorisations dans les politiques AWS gérées. Pour plus d'informations sur les politiques AWS gérées, voir les politiques AWS gérées dans le guide de l'utilisateur IAM.

DynamoDB peut parfois ajouter des autorisations supplémentaires àAWS une politique gérée pour prendre en charge de nouvelles fonctionnalités. Ce type de mise à jour affecte toutes les identités (utilisateurs, groupes et rôles) auxquelles la politique est attachée. Une politique AWS gérée est très susceptible d'être mise à jour lorsqu'une nouvelle fonctionnalité est lancée ou lorsque de nouvelles opérations sont disponibles. DynamoDB ne supprimera pas les autorisations d'AWS une politique gérée. Les mises à jour des politiques n'endommageront donc pas vos autorisations existantes. Pour obtenir la liste complète des politiques AWS gérées, consultez la section stratégies AWS gérées.

AWS politique gérée : Dynamo DBReplication ServiceRolePolicy

Vous ne pouvez pas attacher la politique `DynamoDBReplicationServiceRolePolicy` à vos entités IAM. Cette politique est attachée à un rôle lié au service qui permet à DynamoDB d’effectuer des actions en votre nom. Pour plus d’informations, consultez Utilisation d’IAM avec des tables globales.

Cette politique accorde des autorisations qui permettent au rôle lié au service d’effectuer une réplication de données entre des réplicas de tables globales. Elle accorde également des autorisations administratives pour gérer les réplicas de tables globales en votre nom.

Détails de l’autorisation

Cette politique accorde les autorisations suivantes :

- `dynamodb` : réaliser la réplication des données et gérer les réplicas de tables.
- `application-autoscaling` : extraire et gérer les paramètres d’autoscaling des tables
- `account` : récupérer le statut de la région pour évaluer l’accessibilité des réplicas.

- `iam` : pour créer le rôle lié à un service pour Application Auto Scaling dans le cas où le rôle lié au service n'existerait pas déjà.

La définition de cette politique gérée se trouve ici.

AWS politique gérée : AmazonDynamo DBFull Access_v2

La politique `AmazonDynamoDBFullAccess_v2` limitée accorde des privilèges d'accès spécifiques aux utilisateurs. Vous pouvez associer la politique `AmazonDynamoDBFullAccess_v2` à vos identités IAM. Cette politique accorde un accès administratif aux ressources Amazon DynamoDB et accorde à une identité IAM (telle qu'un utilisateur, un groupe ou un rôle) un accès aux ressources Services AWS auxquelles DynamoDB est intégré afin d'utiliser toutes les fonctionnalités de DynamoDB. L'utilisation de cette politique permet d'accéder à toutes les fonctionnalités de DynamoDB disponibles dans la AWS Management Console.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `Groupes de ressources AWS Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Pour consulter la politique sous `JSON` forme, consultez AmazonDynamoDBFullAccess_v2.

AWS politique gérée : AmazonDynamo DBRead OnlyAccess

Vous pouvez associer la politique `AmazonDynamoDBReadOnlyAccess` à vos identités IAM.

Cette politique accorde un accès en lecture à Amazon DynamoDB.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `Amazon DynamoDB` : fournit un accès en lecture seule à Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)` : fournit un accès en lecture seule à Amazon DynamoDB Accelerator (DAX).
- `Application Auto Scaling` : permet aux principaux de visualiser les configurations à partir d'Application Auto Scaling. Cela est nécessaire pour que les utilisateurs puissent afficher les politiques de mise à l'échelle automatique associées à une table.
- `CloudWatch`— Permet aux principaux de consulter les données métriques et les alarmes configurées dans CloudWatch. Cela est nécessaire pour que les utilisateurs puissent voir la taille de la table facturable et les CloudWatch alarmes configurées pour une table.
- `AWS Data Pipeline`— Permet aux principaux de visualiser AWS Data Pipeline les objets associés.
- `Amazon EC2`— Permet aux principaux de consulter Amazon EC2 VPCs, les sous-réseaux et les groupes de sécurité.
- `IAM` : permet aux principaux d'afficher les rôles IAM.
- `AWS KMS`— Permet aux principaux d'afficher les clés configurées dans AWS KMS. Cela est nécessaire pour que les utilisateurs puissent voir AWS KMS keys ce qu'ils créent et gèrent dans leur compte.
- `Amazon SNS` : permet aux principaux de répertorier les rubriques Amazon SNS et les abonnements par rubrique.
- `Groupes de ressources AWS` : permet aux principaux d'afficher les groupes de ressources et leurs requêtes.
- `Groupes de ressources AWS Tagging` : permet aux principaux de répertorier toutes les ressources balisées ou précédemment balisées d'une région.
- `Kinesis` : permet aux principaux de consulter les descriptions des flux de données Kinesis.
- `Amazon CloudWatch Contributor Insights` : permet aux principaux de consulter les données de séries temporelles collectées par les règles Contributor Insights.

Pour consulter le `JSON` format de la politique, voir AmazonDynamoDBReadOnlyAccess.

Mises à jour DynamoDB des politiques gérées AWS

Ce tableau présente les mises à jour apportées aux politiques de gestion des AWS accès pour DynamoDB.

Modifier	Description	Date de modification
`AmazonDynamoDBFullAccess` : obsolète	Cette politique a été remplacée par une politique limitée nommée `AmazonDynamoDBFullAccess_v2` . Depuis avril 2025, vous ne pouvez plus attacher la politique `AmazonDynamoDBFullAccess` à de nouveaux utilisateurs, groupes ou rôles. Pour de plus amples informations, veuillez consulter AWS politique gérée : AmazonDynamo DBFull Access_v2.	28 avril 2025
Mise à jour de `AmazonDynamoDBReadOnlyAccess` vers une politique existante	`AmazonDynamoDBReadOnlyAccess` a ajouté les autorisations `dynamodb:GetAbacStatus` et `dynamodb:UpdateAbacStatus` . Ces autorisations vous permettent de consulter le statut ABAC et d'activer ABAC pour vous Compte AWS dans la région actuelle.	18 novembre 2024
Mise à jour de `AmazonDynamoDBReadOnlyAccess` vers une politique existante	`AmazonDynamoDBReadOnlyAccess` a ajouté l’autorisation `dynamodb:GetResourcePolicy` . Cette autorisation permet d’accéder aux politiques basées sur les ressources associées aux ressources DynamoDB.	20 mars 2024

Modifier	Description	Date de modification
Mise à jour de `DynamoDBReplicationServiceRolePolicy` vers une politique existante	`DynamoDBReplicationServiceRolePolicy` a ajouté l'autorisation `dynamodb:GetResourcePolicy`. Cette autorisation permet au rôle lié au service de lire les politiques basées sur les ressources associées aux ressources DynamoDB.	15 décembre 2023
Mise à jour de `DynamoDBReplicationServiceRolePolicy` vers une politique existante	`DynamoDBReplicationServiceRolePolicy` a ajouté l'autorisation `account:ListRegions`. Cette autorisation permet au rôle lié au service d'évaluer l'accessibilité des réplicas.	10 mai 2023
`DynamoDBReplicationServiceRolePolicy` ajouté à la liste des politiques gérées	Ajout d'informations sur la politique gérée `DynamoDBReplicationServiceRolePolicy`, qui est utilisée par le rôle lié au service des tables globales DynamoDB.	10 mai 2023
Les tables globales DynamoDB ont commencé à effectuer le suivi des modifications	Les tables globales DynamoDB ont commencé à suivre les modifications apportées à ses politiques gérées.AWS	10 mai 2023

Utilisation de politiques basées sur des ressources pour DynamoDB

DynamoDB prend en charge les politiques basées sur les ressources pour les tables, les index et les flux. Les politiques basées sur les ressources vous permettent de définir les autorisations d'accès en spécifiant qui a accès à chaque ressource et les actions que cette personne est autorisée à effectuer sur chaque ressource.

Vous pouvez attacher une politique basée sur les ressources aux ressources DynamoDB, telles qu'une table ou un flux. Dans cette politique, vous spécifiez les autorisations pour les principaux d'Identity and Access Management (IAM) qui peuvent effectuer des actions spécifiques sur ces ressources DynamoDB. Par exemple, la politique attachée à une table contiendra des autorisations pour accéder à la table et à ses index. Par conséquent, les politiques basées sur les ressources peuvent vous aider à simplifier le contrôle d'accès pour vos tables, index et flux DynamoDB, en définissant des autorisations au niveau de la ressource. La taille maximale d'une politique que vous pouvez attacher à une ressource DynamoDB est de 20 Ko.

L'un des principaux avantages liés à l'utilisation de politiques basées sur les ressources est de simplifier le contrôle d'accès intercompte afin de le fournir aux principaux IAM dans différents Comptes AWS. Pour de plus amples informations, veuillez consulter Politique basées sur les ressources pour l'accès intercompte.

Les politiques basées sur les ressources prennent également en charge les intégrations avec l'analyseur d'accès externe d'IAM Access Analyzer et les fonctionnalités Bloquer l'accès public (BPA). IAM Access Analyzer signale l'accès intercompte aux entités externes spécifiées dans les politiques basées sur les ressources. Il fournit également de la visibilité pour vous aider à affiner les autorisations et à vous conformer au principe du moindre privilège. La fonctionnalité BPA vous aide à empêcher l'accès public à vos tables, index et flux DynamoDB, et elle est automatiquement activée dans les flux de travail de création et de modification de politiques basés sur les ressources.

Rubriques

- Création d'une table avec une politique basée sur les ressources
- Attache d'une politique à une table DynamoDB existante
- Attache d'une politique basée sur les ressources à un flux DynamoDB
- Suppression d'une politique basée sur les ressources d'une table DynamoDB
- Accès intercompte avec des politiques basées sur les ressources dans DynamoDB

- Blocage de l'accès public à l'aide de politiques basées sur les ressources dans DynamoDB
- Opérations d'API DynamoDB prises en charge par des politiques basées sur les ressources
- Autorisation avec des politiques basées sur l'identité IAM et des politiques basées sur les ressources DynamoDB
- Exemples de politiques basées sur les ressources DynamoDB
- Considérations relatives aux politiques basées sur les ressources DynamoDB
- Bonnes pratiques en matière de politiques basées sur les ressources DynamoDB

Création d'une table avec une politique basée sur les ressources

Vous pouvez ajouter une politique basée sur les ressources lorsque vous créez une table à l'aide de la console DynamoDB, de l'CreateTableAPI AWS CLI,AWS du SDK ou d'un modèle.CloudFormation

AWS CLI

L'exemple suivant crée une table nommée à *MusicCollection* l'aide de la `create-table`AWS CLI commande. Cette commande inclut également le paramètre `resource-policy` qui ajoute une politique basée sur les ressources à la table. Cette politique permet *John* à l'utilisateur d'effectuer les actions RestoreTableToPointInTimeGetItem,, et PutItemd'API sur la table.

N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                    },
                    \"Action\": [
```

```
                        \"dynamodb:RestoreTableToPointInTime\",
                        \"dynamodb:GetItem\",
                        \"dynamodb:DescribeTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Sur le tableau de bord, choisissez Créer une table.
3. Dans Détails de la table, saisissez le nom de la table, la clé de partition et les détails de la clé de tri.
4. Pour les Paramètres de la table, choisissez Personnaliser les paramètres.
5. (Facultatif) Spécifiez vos options pour la Classe de table, le Calculateur de capacité, les Paramètres de capacité de lecture/écriture, les Index secondaires, le Chiffrement au repos et la Protection contre la suppression.
6. Dans Politique basée sur les ressources, ajoutez une politique pour définir les autorisations d'accès pour la table et ses index. Dans cette politique, vous spécifiez quels utilisateurs ont accès à ces ressources et les actions qu'ils sont autorisés à effectuer sur chaque ressource. Pour ajouter une politique, effectuez l'une des actions suivantes :

   - Composez ou collez un document de politique JSON. Pour plus de détails sur le langage de politique IAM, consultez Creating policies using the JSON editor dans le Guide de l'utilisateur IAM.

     > **Tip**
     >
     > Pour voir des exemples de politiques basées sur les ressources dans le Guide de développement Amazon DynamoDB, sélectionnez Exemples de politiques.

   - Choisissez Ajouter une nouvelle instruction pour ajouter une nouvelle instruction et saisissez les informations dans les champs fournis. Répétez l'opération pour autant d'instructions que vous souhaitez ajouter.

> **Important**
>
> Veillez à résoudre les avertissements de sécurité, les erreurs ou les suggestions avant d'enregistrer votre politique.

L'exemple de politique IAM suivant permet *John* à l'utilisateur d'effectuer les actions RestoreTableToPointInTimeGetItem,, et PutItemAPI sur la table*MusicCollection*.

N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Facultatif) Choisissez Preview external access (Aperçu de l'accès externe) dans le coin inférieur droit pour avoir un aperçu de la façon dont votre nouvelle politique affecte l'accès public et l'accès intercompte à votre ressource. Avant d'enregistrer votre stratégie, vous pouvez vérifier si elle introduit de nouveaux résultats IAM Access Analyzer ou si elle résout les résultats existants. Si vous ne voyez pas d'analyseur actif, choisissez Go to Access Analyzer (Accédez à

l'analyseur d'accès) pour créer un analyseur de compte dans l'analyseur d'accès IAM. Pour plus d'informations, consultez Prévisualiser l'accès.

8. Choisissez Créer un tableau.

AWS CloudFormation modèle

Using the AWS::DynamoDB::Table resource

Le CloudFormation modèle suivant crée une table avec un flux à l'aide de la ressource AWS::DynamoDB : :Table. Ce modèle inclut également des politiques basées sur les ressources, associées à la fois à la table et au flux.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    }
                ],
                "BillingMode": "PROVISIONED",
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "StreamSpecification": {
                  "StreamViewType": "OLD_IMAGE",
                  "ResourcePolicy": {
                    "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [
                        {
```

```
                            "Principal": {
                                "AWS": "arn:aws:iam::111122223333:user/John"
                            },
                            "Effect": "Allow",
                            "Action": [
                                "dynamodb:GetRecords",
                                "dynamodb:GetShardIterator",
                                "dynamodb:DescribeStream"
                            ],
                            "Resource": "*"
                        }
                      ]
                    }
                  }
                },
                "TableName": "MusicCollection",
                "ResourcePolicy": {
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Principal": {
                                    "AWS": [
                                        "arn:aws:iam::111122223333:user/John"
                                    ]
                                },
                                "Effect": "Allow",
                                "Action": "dynamodb:GetItem",
                                "Resource": "*"
                            }
                        ]
                    }
                }
            }

        }
    }
}
```

Using the AWS::DynamoDB::GlobalTable resource

Le CloudFormation modèle suivant crée une table avec la AWS GlobalTable ressource ::DynamoDB : : et attache une politique basée sur les ressources à la table et à son flux.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
```

```
                                            "AWS":
 "arn:aws:iam::111122223333:user/John"
                                        },
                                        "Effect": "Allow",
                                        "Action": [
                                            "dynamodb:GetRecords",
                                            "dynamodb:GetShardIterator",
                                            "dynamodb:DescribeStream"
                                        ],
                                        "Resource": "*"
                                    }]
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

Attache d'une politique à une table DynamoDB existante

Vous pouvez associer une stratégie basée sur les ressources à une table existante ou modifier une stratégie existante à l'aide de la console DynamoDB, de l'PutResourcePolicyAPI AWS CLI,AWS du SDK ou d'un modèle.CloudFormation

AWS CLI exemple pour joindre une nouvelle politique

L'exemple de stratégie IAM suivant utilise la `put-resource-policy`AWS CLI commande pour associer une stratégie basée sur les ressources à une table existante. Cet exemple permet *John* à l'utilisateur d'effectuer les actions GetItemPutItem, UpdateItem, et d'UpdateTableAPI sur une table existante nommée*MusicCollection*.

N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
```

```
            {
                  \"Effect\": \"Allow\",
                  \"Principal\": {
                      \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                  },
                  \"Action\": [
                      \"dynamodb:GetItem\",
                      \"dynamodb:PutItem\",
                      \"dynamodb:UpdateItem\",
                      \"dynamodb:UpdateTable\"
                  ],
                  \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS CLI exemple de mise à jour conditionnelle d'une politique existante

Pour mettre à jour de manière conditionnelle la politique basée sur les ressources existante d'une table, vous pouvez utiliser le paramètre facultatif `expected-revision-id`. L'exemple suivant met à jour la politique uniquement si elle existe dans DynamoDB et si son ID de révision actuel correspond au paramètre `expected-revision-id` fourni.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                  \"Effect\": \"Allow\",
                  \"Principal\": {
                      \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                  },
                  \"Action\": [
                      \"dynamodb:GetItem\",
                      \"dynamodb:UpdateItem\",
                      \"dynamodb:UpdateTable\"
                  ],
                  \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
```

```
            }
        ]
    }"
```

AWS Management Console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Sur le tableau de bord, choisissez une table existante.
3. Accédez à l'onglet Autorisations, puis choisissez Créer une politique pour la table.
4. Dans l'éditeur de politique basée sur les ressources, ajoutez la politique que vous souhaitez associer et choisissez Créer une politique.

   L'exemple de politique IAM suivant permet *John* à l'utilisateur d'effectuer les actions GetItemPutItem,, UpdateItem, et d'UpdateTableAPI sur une table existante nomméeMusicCollection.

   N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
```

```
}
```

AWS SDK for Java 2.x

L'exemple de politique IAM suivant utilise la méthode `putResourcePolicy` pour attacher une politique basée sur les ressources à une table existante. Cette politique permet à un utilisateur d'exécuter l'action d'[GetItem]API sur une table existante.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
 For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowed AWS Principal - Allowed AWS principal ARN that the example
 policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
```

```
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
 table with ARN " +
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }

    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
 allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return "";
    }

    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +,
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
```

```
                "        }\n" +
                "    ]\n" +
                "}";
    }
}
```

Attache d'une politique basée sur les ressources à un flux DynamoDB

Vous pouvez associer une stratégie basée sur les ressources au flux d'une table existante ou modifier une stratégie existante à l'aide de la console DynamoDB, de l'PutResourcePolicyAPI AWS CLI,AWS du SDK ou d'un modèle.CloudFormation

> **Note**
>
> Vous ne pouvez pas associer de politique à un flux lorsque vous le créez à l'aide du CreateTableou UpdateTable APIs. Vous pouvez toutefois modifier ou supprimer une politique après la suppression d'une table. Vous pouvez également modifier ou supprimer la politique d'un flux désactivé.

AWS CLI

L'exemple de stratégie IAM suivant utilise la `put-resource-policy`AWS CLI commande pour associer une stratégie basée sur les ressources au flux d'une table nommée. *`MusicCollection`* Cet exemple permet *`John`* à l'utilisateur d'effectuer les actions GetRecordsGetShardIterator,, et DescribeStreamAPI sur le flux.

N'oubliez pas de remplacer le *`italicized`* texte par les informations spécifiques à votre ressource.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
```

```
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
                        \"dynamodb:GetShardIterator\",
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

AWS Management Console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le tableau de bord de la console DynamoDB, choisissez Tables, puis sélectionnez une table existante.

   Assurez-vous que les flux sont activés dans la table que vous sélectionnez. Pour en savoir plus sur l'activation de flux pour une table, consultez Activation d'un flux.
3. Sélectionnez l'onglet Autorisations.
4. Dans Politique basée sur les ressources pour le flux actif, choisissez Créer une politique pour le flux.
5. Dans l'éditeur Politique basée sur les ressources, ajoutez une politique pour définir les autorisations d'accès pour le flux. Dans cette politique, vous spécifiez quels utilisateurs ont accès au flux et les actions qu'ils sont autorisés à effectuer sur le flux. Pour ajouter une politique, effectuez l'une des actions suivantes :

   - Composez ou collez un document de politique JSON. Pour plus de détails sur le langage de politique IAM, consultez Creating policies using the JSON editor dans le Guide de l'utilisateur IAM.

   > **Tip**
   >
   > Pour voir des exemples de politiques basées sur les ressources dans le Guide de développement Amazon DynamoDB, sélectionnez Exemples de politiques.

- Choisissez Ajouter une nouvelle instruction pour ajouter une nouvelle instruction et saisissez les informations dans les champs fournis. Répétez l'opération pour autant d'instructions que vous souhaitez ajouter.

> **Important**
>
> Veillez à résoudre les avertissements de sécurité, les erreurs ou les suggestions avant d'enregistrer votre politique.

6. (Facultatif) Choisissez Preview external access (Aperçu de l'accès externe) dans le coin inférieur droit pour avoir un aperçu de la façon dont votre nouvelle politique affecte l'accès public et l'accès intercompte à votre ressource. Avant d'enregistrer votre stratégie, vous pouvez vérifier si elle introduit de nouveaux résultats IAM Access Analyzer ou si elle résout les résultats existants. Si vous ne voyez pas d'analyseur actif, choisissez Go to Access Analyzer (Accédez à l'analyseur d'accès) pour créer un analyseur de compte dans l'analyseur d'accès IAM. Pour plus d'informations, consultez Prévisualiser l'accès.
7. Choisissez Create Policy (Créer une politique).

L'exemple de stratégie IAM suivant attache une stratégie basée sur les ressources au flux d'une table nommée. *MusicCollection* Cet exemple permet *John* à l'utilisateur d'effectuer les actions GetRecordsGetShardIterator,, et DescribeStreamAPI sur le flux.

N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
```

```
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Suppression d'une politique basée sur les ressources d'une table DynamoDB

Vous pouvez supprimer une politique basée sur les ressources d'une table existante à l'aide de la console DynamoDB, de l'DeleteResourcePolicyAPI AWS CLI,AWS du SDK ou d'un modèle.CloudFormation

AWS CLI

L'exemple suivant utilise la `delete-resource-policy`AWS CLI commande pour supprimer une politique basée sur les ressources d'une table nommée. *MusicCollection*

N'oubliez pas de remplacer le *italicized* texte par les informations spécifiques à votre ressource.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le tableau de bord de la console DynamoDB, choisissez Tables, puis sélectionnez une table existante.
3. Choisissez Autorisations.
4. Dans le menu déroulant Gérer la politique, choisissez Supprimer la politique.
5. Dans la boîte de dialogue Supprimer la politique basée sur les ressources pour la table, saisissez **`confirm`** pour confirmer la suppression.
6. Sélectionnez Delete (Supprimer).

Accès intercompte avec des politiques basées sur les ressources dans DynamoDB

À l'aide d'une politique basée sur les ressources, vous pouvez fournir un accès intercompte à des ressources disponibles dans différents Comptes AWS. Tous les accès entre comptes autorisés par les politiques basées sur les ressources seront signalés par le biais des résultats d'accès externes d'IAM Access Analyzer si vous disposez d'un analyseur identique à la ressource.Région AWS IAM Access Analyzer exécute des vérifications de politiques pour valider votre politique par rapport à la grammaire de politique et aux bonnes pratiques IAM. Ces vérifications génèrent des résultats et fournissent des recommandations exploitables pour vous aider à créer des stratégies fonctionnelles et conformes aux bonnes pratiques en matière de sécurité. Vous pouvez consulter les résultats actifs d'IAM Access Analyzer dans l'onglet Autorisations de la console DynamoDB.

Pour en savoir plus sur la validation des politiques à l'aide d'IAM Access Analyzer, consultez Validation de la politique de l'IAM Access Analyzer dans le Guide de l'utilisateur IAM. Pour afficher la liste des avertissements, erreurs et suggestions renvoyés par IAM Access Analyzer, consultez la Référence de vérification de politique IAM Access Analyzer.

Pour GetItemautoriser un utilisateur A du compte A à accéder à une table B du compte B, effectuez les opérations suivantes :

1. Attachez à la table B une politique basée sur les ressources qui autorise l'utilisateur A à effectuer l'action `GetItem`.
2. Attachez une politique basée sur l'identité qui autorise l'utilisateur A à effectuer l'action `GetItem` sur la table B.

À l'aide de l'option Aperçu de l'accès externe disponible dans la console DynamoDB, vous pouvez prévisualiser la façon dont votre nouvelle politique affecte l'accès public et intercompte à votre ressource. Avant d'enregistrer votre stratégie, vous pouvez vérifier si elle introduit de nouveaux résultats IAM Access Analyzer ou si elle résout les résultats existants. Si vous ne voyez pas d'analyseur actif, choisissez Go to Access Analyzer (Accédez à l'analyseur d'accès) pour créer un analyseur de compte dans l'analyseur d'accès IAM. Pour plus d'informations, consultez Prévisualiser l'accès.

Le paramètre de nom de table dans le plan de données et le plan de contrôle DynamoDB APIs accepte le nom de ressource Amazon (ARN) complet de la table afin de prendre en charge les opérations entre comptes. Si vous fournissez uniquement le paramètre de nom de table au lieu

d'un ARN complet, l'opération d'API sera exécutée sur la table du compte auquel appartient le demandeur. Pour obtenir un exemple de politique qui utilise l'accès intercompte, consultez Politique basées sur les ressources pour l'accès intercompte.

Le compte du propriétaire de la ressource sera débité même lorsque le principal d'un autre compte lit ou écrit dans la table DynamoDB du compte du propriétaire. Si la table dispose d'un débit provisionné, la somme de toutes les demandes provenant des comptes propriétaires et des demandeurs des autres comptes déterminera si la demande sera limitée (si le dimensionnement automatique est désactivé) ou redimensionnée si le dimensionnement automatique est activé. up/down

Les demandes seront enregistrées dans les CloudTrail journaux des comptes propriétaire et demandeur afin que chacun des deux comptes puisse savoir quel compte a accédé à quelles données.

> **Note**
>
> L'accès entre comptes du plan de contrôle APIs a une limite de transactions par seconde (TPS) inférieure à 500 demandes.

Blocage de l'accès public à l'aide de politiques basées sur les ressources dans DynamoDB

Bloquer l'accès public (BPA) est une fonctionnalité qui identifie et empêche l'attachement de politiques basées sur les ressources qui accordent un accès public à vos tables, index ou flux DynamoDB sur vos comptes Amazon Web Services (AWS). Avec BPA, vous pouvez empêcher l'accès public à vos ressources DynamoDB. BPA effectue des vérifications lors de la création ou de la modification d'une politique basée sur les ressources et contribue à améliorer votre niveau de sécurité avec DynamoDB.

BPA utilise un raisonnement automatisé pour analyser l'accès accordé par votre politique basée sur les ressources, et vous alerte si de telles autorisations sont détectées au moment de l'administration d'une politique basée sur les ressources. L'analyse vérifie l'accès à toutes les déclarations de politique basées sur les ressources, aux actions et à l'ensemble de clés de condition utilisées dans vos politiques.

> **Important**
>
> BPA permet de protéger vos ressources en empêchant l'octroi de l'accès public par le biais de politiques basées sur les ressources directement associées à vos ressources DynamoDB, telles que les tables, les index et les flux. En plus d'utiliser BPA, examinez attentivement les politiques suivantes pour vous assurer qu'elles n'accordent pas d'accès public :
>
> - Politiques basées sur l'identité associées aux AWS principaux associés (par exemple, rôles IAM)
> - Politiques basées sur les AWS ressources associées (par exemple, clés AWS Key Management Service(KMS))

Vous devez vous assurer que le principal n'inclut aucune entrée * ou que l'une des clés de condition spécifiées limite l'accès des principaux à la ressource. Si la politique basée sur les ressources accorde un accès public à votre table, à vos index ou à votre flux,Comptes AWS DynamoDB vous empêchera de créer ou de modifier la politique jusqu'à ce que la spécification contenue dans la stratégie soit corrigée et considérée comme non publique.

Vous pouvez rendre une politique non publique en spécifiant un ou plusieurs principaux à l'intérieur du bloc `Principal`. L'exemple de politique basée sur les ressources suivant bloque l'accès public en spécifiant deux principaux.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

Les politiques qui restreignent l'accès en spécifiant certaines clés de condition ne sont pas non plus considérées comme publiques. Parallèlement à l'évaluation du principal spécifié dans la politique basée sur les ressources, les clés de condition fiables suivantes sont utilisées pour terminer l'évaluation d'une politique basée sur les ressources pour un accès non public :

- aws:PrincipalAccount
- aws:PrincipalArn
- aws:PrincipalOrgID
- aws:PrincipalOrgPaths
- aws:SourceAccount
- aws:SourceArn
- aws:SourceVpc
- aws:SourceVpce
- aws:UserId
- aws:PrincipalServiceName
- aws:PrincipalServiceNamesList
- aws:PrincipalIsAWSService
- aws:Ec2InstanceSourceVpc
- aws:SourceOrgID
- aws:SourceOrgPaths

En outre, pour qu'une politique basée sur les ressources ne soit pas publique, les valeurs d'Amazon Resource Name (ARN) et les clés de chaîne ne doivent pas contenir de caractères génériques ni de variables. Si votre politique basée sur les ressources utilise la clé `aws:PrincipalIsAWSService`, vous devez vous assurer que vous avez défini la valeur de la clé sur true.

La politique suivante limite l'accès à l'utilisateur `John` dans le compte spécifié. La condition impose une contrainte à `Principal` et ne doit pas être considérée comme publique.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
```

```
  }
}
```

L'exemple suivant d'une politique basée sur des ressources non publique limite l'utilisation de `sourceVPC` avec l'opérateur `StringEquals`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Opérations d'API DynamoDB prises en charge par des politiques basées sur les ressources

Cette rubrique répertorie les opérations d'API prises en charge par les politiques basées sur les ressources. Toutefois, pour l'accès entre comptes, vous ne pouvez utiliser qu'un certain ensemble de APIs DynamoDB via des politiques basées sur les ressources. Vous ne pouvez pas attacher de politiques basées sur les ressources à des types de ressources, tels que les sauvegardes et les importations. Les actions IAM, qui correspondent à l' APIs opération sur ces types de ressources, sont exclues des actions IAM prises en charge dans les politiques basées sur les ressources. Parce

que les administrateurs de tables configurent les paramètres internes des tables au sein d'un même compte APIs, tels que UpdateTimeToLiveet DisableKinesisStreamingDestination, ne prennent pas en charge l'accès entre comptes via des politiques basées sur les ressources.

Le plan de données et le plan de contrôle APIs DynamoDB qui prennent en charge l'accès entre comptes prennent également en charge la surcharge des noms de table, ce qui vous permet de spécifier l'ARN de la table au lieu du nom de la table. Vous pouvez spécifier l'ARN de la table dans le `TableName` paramètre de ceux-ci APIs. Cependant, ils ne sont pas tous compatibles avec APIs l'accès entre comptes.

Rubriques

- Opérations de l’API du plan de données
- Opérations d’une API PartiQL
- Opérations de l’API du plan de contrôle
- Opérations API des tables globales de la version 2019.11.21 (actuelle)
- Opérations API des tables globales de la version 2017.11.29 (ancienne)
- Opérations d’API de balises
- Opérations de l’API de sauvegarde et restauration
- Opérations d'API continues Backup/Restore (PITR)
- Opérations d’API Contributor Insights
- Opérations d’API d’exportation
- Opérations d’API d’importation
- Opérations de l’API Amazon Kinesis Data Streams
- Opérations d’API de politique basée sur les ressources
- Time-to-Live Opérations d'API
- Autres opérations d’API
- Opérations liées à l’API DynamoDB Streams

Opérations de l’API du plan de données

Le tableau suivant répertorie la prise en charge au niveau de l’API fournie par les opérations d’API du plan de données, pour les politiques basées sur les ressources et l’accès intercompte.

Plan de données - Tables/in dexes APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DeleteItem	Oui	Oui
GetItem	Oui	Oui
PutItem	Oui	Oui
Interrogation	Oui	Oui
Analyser	Oui	Oui
UpdateItem	Oui	Oui
TransactGetItems	Oui	Oui
TransactWriteItems	Oui	Oui
BatchGetItem	Oui	Oui
BatchWriteItem	Oui	Oui

Opérations d'une API PartiQL

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API PartiQL pour les politiques basées sur les ressources et l'accès intercompte.

PartiQL APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
BatchExecuteStatement	Oui	Non
ExecuteStatement	Oui	Non
ExecuteTransaction	Oui	Non

Opérations de l'API du plan de contrôle

Le tableau suivant répertorie la prise en charge au niveau de l'API fournie par les opérations d'API du plan de contrôle, pour les politiques basées sur les ressources et l'accès intercompte.

Plan de contrôle - Tables APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
CreateTable	Non	Non
DeleteTable	Oui	Oui
DescribeTable	Oui	Oui
UpdateTable	Oui	Oui

Opérations API des tables globales de la version 2019.11.21 (actuelle)

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API des tables globales version 2019.11.21 (actuelle) pour les politiques basées sur les ressources et l'accès intercompte.

Version 2019.11.21 (actuelle) des tables globales APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeTableReplicaAutoScaling	Oui	Non
UpdateTableReplicaAutoScaling	Oui	Non

Opérations API des tables globales de la version 2017.11.29 (ancienne)

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API des Tables globales de la version 2017.11.29 (ancienne) pour les politiques basées sur les ressources et l'accès intercompte.

Tables globales de la version 2017.11.29 (Legacy) APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
CreateGlobalTable	Non	Non
DescribeGlobalTable	Non	Non
DescribeGlobalTableSettings	Non	Non
ListGlobalTables	Non	Non
UpdateGlobalTable	Non	Non
UpdateGlobalTableSettings	Non	Non

Opérations d'API de balises

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API liées aux balises pour les politiques basées sur les ressources et l'accès intercompte.

Balises APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
ListTagsOfResource	Oui	Oui
TagResource	Oui	Oui
UntagResource	Oui	Oui

Opérations de l'API de sauvegarde et restauration

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API liées à la sauvegarde et à la restauration pour les politiques basées sur les ressources et l'accès intercompte.

Backup et restauration APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
CreateBackup	Oui	Non

Backup et restauration APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeBackup	Non	Non
DeleteBackup	Non	Non
RestoreTableFromBackup	Non	Non

Opérations d'API continues Backup/Restore (PITR)

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API liées à Continuous Backup/Restore (PITR) pour les politiques basées sur les ressources et l'accès entre comptes.

Continu Backup/Restore (PITR) APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeContinuousBackups	Oui	Non
RestoreTableToPointInTime	Oui	Non
UpdateContinuousBackups	Oui	Non

Opérations d’API Contributor Insights

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations d'API liées à Continuous Backup/Restore (PITR) pour les politiques basées sur les ressources et l'accès entre comptes.

Informations sur les contributeurs APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeContributorInsights	Oui	Non
ListContributorInsights	Non	Non

Informations sur les contributeurs APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
UpdateContributorInsights	Oui	Non

Opérations d'API d'exportation

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations de l'API d'exportation pour les politiques basées sur les ressources et l'accès intercompte.

Exporter APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeExport	Non	Non
ExportTableToPointInTime	Oui	Non
ListExports	Non	Non

Opérations d'API d'importation

Le tableau suivant répertorie le support au niveau de l'API fourni par les opérations de l'API d'importation pour les politiques basées sur les ressources et l'accès intercompte.

Importer APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeImport	Non	Non
ImportTable	Non	Non
ListImports	Non	Non

Opérations de l'API Amazon Kinesis Data Streams

Le tableau suivant répertorie la prise en charge au niveau de l'API fournie par les opérations de l'API Kinesis Data Streams, pour les politiques basées sur les ressources et l'accès intercompte.

Kinésis APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeKinesisStreamingDestination	Oui	Non
DisableKinesisStreamingDestination	Oui	Non
EnableKinesisStreamingDestination	Oui	Non
UpdateKinesisStreamingDestination	Oui	Non

Opérations d’API de politique basée sur les ressources

Le tableau suivant répertorie le support au niveau de l’API fourni par les opérations de l’API de politique basée sur les ressources pour ces politiques et l’accès intercompte.

Politique basée sur les ressources APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
GetResourcePolicy	Oui	Non
PutResourcePolicy	Oui	Non
DeleteResourcePolicy	Oui	Non

Time-to-Live Opérations d'API

Le tableau suivant répertorie le support au niveau de l’API fourni par les opérations de l’API de durée de vie (TTL) pour les politiques basées sur les ressources et l’accès intercompte.

TTL APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeTimeToLive	Oui	Non
UpdateTimeToLive	Oui	Non

Autres opérations d'API

Le tableau suivant répertorie le support au niveau de l'API fourni par les autres opérations d'API diverses pour les politiques basées sur les ressources et l'accès intercompte.

Autres APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeLimits	Non	Non
DescribeEndpoints	Non	Non
ListBackups	Non	Non
ListTables	Non	Non

Opérations liées à l'API DynamoDB Streams

Le tableau suivant répertorie la prise en charge par APIs DynamoDB Streams au niveau de l'API pour les politiques basées sur les ressources et l'accès entre comptes.

Streams DynamoDB APIs	Prise en charge de politique basée sur les ressources	Prise en charge intercompte
DescribeStream	Oui	Oui
GetRecords	Oui	Oui
GetShardIterator	Oui	Oui
ListStreams	Non	Non

Autorisation avec des politiques basées sur l'identité IAM et des politiques basées sur les ressources DynamoDB

Les politiques basées sur l'identité sont attachées à une identité, comme des utilisateurs, des groupes d'utilisateurs et des rôles IAM. Ces documents de politique IAM contrôlent les actions que peut exécuter une identité, sur quelles ressources et dans quelles conditions. Les politiques basées sur une identité peuvent être gérées ou en ligne.

Les politiques basées sur les ressources sont des documents de politique IAM que vous attachez à une ressource, telle qu'une table DynamoDB. Ces politiques accordent au principal spécifié l'autorisation d'effectuer des actions spécifiques sur cette ressource et définit sous quelles conditions cela s'applique. Par exemple, la politique basée sur les ressources pour une table DynamoDB inclut également l'index associé à la table. Les politiques basées sur les ressources sont des politiques en ligne. Il ne s'agit pas de politiques gérées basées sur les ressources.

Pour plus d'informations sur ces politiques, consultez Politiques basées sur l'identité et Politiques basées sur une ressource dans le Guide de l'utilisateur IAM.

Si le principal IAM provient du même compte que le propriétaire de la ressource, une politique basée sur les ressources est suffisante pour spécifier les autorisations d'accès à la ressource. Vous pouvez toujours choisir d'avoir une politique basée sur l'identité IAM avec une politique basée sur les ressources. Pour l'accès intercompte, vous devez autoriser explicitement l'accès dans les politiques d'identité et de ressources, comme spécifié dans Accès intercompte avec des politiques basées sur les ressources dans DynamoDB. Lorsque vous utilisez les deux types de politiques, une politique est évaluée comme décrit dans Déterminer si une demande est autorisée ou rejetée dans un compte.

Exemples de politiques basées sur les ressources DynamoDB

Lorsque vous spécifiez un ARN dans le champ `Resource` d'une politique basée sur les ressources, la politique ne prend effet que si l'ARN spécifié correspond à celui de la ressource DynamoDB à laquelle il est attaché.

> **Note**
>
> N'oubliez pas de remplacer le *`italicized`* texte par les informations spécifiques à votre ressource.

Politique basée sur les ressources pour une table

La politique basée sur les ressources suivante, attachée à une table DynamoDB nommée*MusicCollection*, donne aux utilisateurs IAM l'*Jane*autorisation d'effectuer *John* des actions sur la ressource. GetItemBatchGetItem*MusicCollection*

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

Politique basée sur les ressources pour un flux

La politique basée sur les ressources suivante attachée à un flux DynamoDB nommé `2024-02-12T18:57:26.492` donne aux utilisateurs IAM l'*Jane*autorisation d'effectuer GetRecordsdes *John* actions d'API sur la ressource. GetShardIteratorDescribeStream`2024-02-12T18:57:26.492`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Politique d’accès basée sur les ressources pour effectuer toutes les actions sur les ressources spécifiées

Pour permettre à un utilisateur d’effectuer toutes les actions sur une table et tous les index associés à une table, vous pouvez utiliser un caractère générique (*) pour représenter les actions et les ressources associées à la table. L’utilisation d’un caractère générique pour les ressources permettra à l’utilisateur d’accéder à la table DynamoDB et à tous ses index associés, y compris ceux qui n’ont pas encore été créés. Par exemple, la politique suivante *John* autorisera l'utilisateur à effectuer des actions sur la *MusicCollection* table et sur tous ses index, y compris les index qui seront créés dans le futur.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
      ]
    }
  ]
}
```

Politique basées sur les ressources pour l'accès intercompte

Vous pouvez spécifier des autorisations pour une identité IAM intercompte afin d'accéder aux ressources DynamoDB. Par exemple, vous pouvez avoir besoin qu'un utilisateur d'un compte approuvé bénéficie d'un accès au contenu de votre table, à condition qu'il n'accède qu'à des éléments et à des attributs spécifiques de ces éléments. La politique suivante autorise l'accès à un utilisateur à *John* partir d'un Compte AWS identifiant fiable *111111111111* pour accéder aux données d'une table dans un compte à *123456789012* l'aide de l'GetItemAPI. La politique garantit que l'utilisateur ne peut accéder qu'aux éléments dotés d'une clé primaire *Jane* et qu'il ne peut récupérer que les attributs `ArtistSongTitle`, mais aucun autre attribut.

> **Important**
>
> Si vous ne spécifiez pas la condition `SPECIFIC_ATTRIBUTES`, vous verrez tous les attributs des articles renvoyés.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

Outre la politique basée sur les ressources précédente, la politique basée sur l'identité attachée à l'utilisateur doit *John* également autoriser l'action de l'`GetItem`API pour que l'accès entre comptes fonctionne. Voici un exemple de politique basée sur l'identité que vous devez associer à l'utilisateur. *John*

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

L'utilisateur John peut faire une `GetItem` demande en spécifiant l'ARN de la table dans le `table-name` paramètre d'accès à la table *MusicCollection* dans le compte*123456789012*.

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

Politique basée sur les ressources avec conditions d’adresse IP

Vous pouvez appliquer une condition pour restreindre les adresses IP sources, les clouds privés virtuels (VPCs) et les points de terminaison VPC (VPCE). Vous pouvez spécifier des autorisations en fonction des adresses sources de la demande d’origine. Par exemple, vous pouvez autoriser un utilisateur à accéder aux ressources DynamoDB uniquement si elles sont accessibles à partir d’une

source IP spécifique, telle qu'un point de terminaison VPN d'entreprise. Spécifiez ces adresses IP dans la déclaration `Condition`.

L'exemple suivant permet à l'utilisateur d'*John*accéder à n'importe quelle ressource DynamoDB lorsque les IPs sources sont et. `54.240.143.0/24` `2001:DB8:1234:5678::/64`

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

Vous pouvez également refuser tout accès aux ressources DynamoDB, sauf lorsque la source est un point de terminaison VPC spécifique, par exemple. *vpce-1a2b3c4d*

> **Important**
>
> Lorsque vous utilisez DAX avec des tables DynamoDB dotées de politiques de ressources basées sur l'adresse IP IPv6 dans des environnements uniquement, vous devez configurer des règles d'accès supplémentaires. Si votre politique de ressources restreint l'accès à

l'espace d' IPv4 adressage `0.0.0.0/0` des tables, vous devez autoriser l'accès au rôle IAM associé à votre cluster DAX. Ajoutez une `ArnNotEquals` condition à votre politique pour garantir que DAX conserve l'accès à vos tables DynamoDB. Pour plus d'informations, voir DAX et. IPv6

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Politique basée sur les ressources utilisant un rôle IAM

Vous pouvez également spécifier un rôle de service IAM dans la politique basée sur les ressources. Les entités IAM qui assument ce rôle sont limitées par les actions autorisées spécifiées pour le rôle et par l'ensemble de ressources spécifique dans le cadre de la politique basée sur les ressources.

L'exemple suivant permet à une entité IAM d'effectuer toutes les actions DynamoDB sur les ressources DynamoDB et DynamoDB. *MusicCollection* *MusicCollection*

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

Considérations relatives aux politiques basées sur les ressources DynamoDB

Lorsque vous définissez des politiques basées sur les ressources pour vos ressources DynamoDB, les considérations suivantes s'appliquent :

Considérations générales

- La taille maximale prise en charge pour un document de politique basé sur les ressources est de 20 Ko. DynamoDB compte les espaces blancs lors du calcul de la taille d'une politique au regard de cette limite.
- Les mises à jour ultérieures d'une politique pour une ressource donnée sont bloquées pendant 15 secondes après une mise à jour réussie de la politique pour la même ressource.
- Vous ne pouvez actuellement attacher une politique basée sur les ressources qu'aux flux existants. Vous ne pouvez pas attacher de politique à un flux lors de sa création.

Considérations relatives aux tables globales

- Les politiques basées sur les ressources ne sont pas prises en charge pour les réplicas de Table globale version 2017.11.29 (ancienne).
- Dans une politique basée sur les ressources, si l'action permettant à un rôle DynamoDB lié à un service (SLR) de répliquer les données d'une table globale est refusée, l'ajout ou la suppression d'un réplica échouera avec une erreur.

- La AWS GlobalTable ressource ::DynamoDB : : ne prend pas en charge la création d'une réplique et l'ajout d'une politique basée sur les ressources à cette réplique dans la même mise à jour de pile dans des régions autres que la région où vous déployez la mise à jour de pile.

Considérations relatives à l’accès intercompte

- L'accès entre comptes à l'aide de politiques basées sur les ressources ne prend pas en charge les tables chiffrées avec des clés AWS gérées, car vous ne pouvez pas accorder d'accès entre comptes à la AWS politique KMS gérée.

CloudFormation considérations

- Les politiques basées sur les ressources ne prennent pas en charge la détection des écarts. Si vous mettez à jour une politique basée sur les ressources en dehors du modèle de AWS CloudFormation pile, vous devrez mettre à jour la CloudFormation pile avec les modifications.
- Les politiques basées sur les ressources ne prennent pas en charge les modifications hors bande. Si vous ajoutez, mettez à jour ou supprimez une politique en dehors du CloudFormation modèle, la modification ne sera pas remplacée si aucune modification n'est apportée à la politique dans le modèle.

  Supposons, par exemple, que votre modèle contienne une politique basée sur les ressources que vous mettez à jour ultérieurement en dehors du modèle. Si vous n’apportez aucune modification à la politique du modèle, la politique mise à jour dans DynamoDB ne sera pas synchronisée avec celle du modèle.

  Inversement, supposons que votre modèle ne contienne pas de politique basée sur les ressources, mais que vous ajoutiez une politique en dehors du modèle. Cette politique ne sera pas supprimée de DynamoDB tant que vous ne l’ajoutez pas au modèle. Lorsque vous ajoutez une politique au modèle et que vous mettez à jour la pile, la politique existante dans DynamoDB est mise à jour pour correspondre à celle définie dans le modèle.

Bonnes pratiques en matière de politiques basées sur les ressources DynamoDB

Cette rubrique décrit les bonnes pratiques pour définir les autorisations d’accès pour vos ressources DynamoDB et les actions autorisées sur ces ressources.

Simplification du contrôle d’accès aux ressources DynamoDB

Si Gestion des identités et des accès AWS les principaux qui ont besoin d'accéder à une ressource DynamoDB font partie de la Compte AWS même entité que le propriétaire de la ressource, aucune politique basée sur l'identité IAM n'est requise pour chaque principal. Une politique basée sur les ressources qui est attachée aux ressources données suffit. Ce type de configuration simplifie le contrôle d’accès.

Protégez vos ressources DynamoDB avec les politiques basées sur les ressources

Pour toutes les tables et tous les flux DynamoDB, créez des politiques basées sur les ressources afin d’appliquer le contrôle d’accès à ces ressources. Les politiques basées sur les ressources vous permettent de centraliser les autorisations au niveau des ressources, de simplifier le contrôle d’accès aux tables, aux index et aux flux DynamoDB, et de réduire les frais d’administration. Si aucune politique basée sur les ressources n’est spécifiée pour une table ou un flux, l’accès à la table ou au flux sera implicitement refusé, sauf si les politiques basées sur l’identité associées aux principaux IAM autorisent l’accès.

Accorder les autorisations de moindre privilège

Lorsque vous définissez des autorisations avec des politiques basées sur les ressources DynamoDB, accordez uniquement les autorisations nécessaires à l’exécution d’une action. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Vous pouvez commencer par des autorisations étendues tout en explorant les autorisations requises pour votre charge de travail ou votre cas d’utilisation. Au fur et à mesure que votre cas d’utilisation évolue, vous pouvez réduire les autorisations que vous accordez pour obtenir le moindre privilège.

Analyse de l’activité d’accès intercompte pour générer des politiques de moindre privilège

IAM Access Analyzer signale l’accès intercompte aux entités externes spécifiées dans les politiques basées sur les ressources, et fournit une visibilité qui vous aide à affiner les autorisations et à vous conformer au moindre privilège. Pour plus d’informations sur la génération de politiques, consultez IAM Access Analyzer policy generation.

Utilisation d'IAM Access Analyzer pour générer des politiques de moindre privilège

Pour accorder uniquement les autorisations nécessaires à l'exécution d'une seule tâche, vous pouvez générer des politiques en fonction de votre activité d'accès connectée AWS CloudTrail. IAM Access Analyzer analyse les services et les actions utilisés par vos politiques.

Utilisation du contrôle d'accès par attributs avec DynamoDB

Le contrôle d'accès basé sur les attributs (ABAC) est une stratégie d'autorisation qui définit les autorisations d'accès en fonction des conditions des balises dans vos politiques basées sur l'identité ou d'autres politiques, telles que les politiques basées sur les ressources et AWS les politiques IAM de l'organisation. Vous pouvez associer des balises aux tables DynamoDB, qui sont ensuite évaluées par rapport aux conditions basées sur les balises. Les index associés à une table héritent des balises que vous ajoutez à la table. Vous pouvez ajouter jusqu'à 50 balises par table DynamoDB. La taille maximale prise en charge pour toutes les balises d'une table est de 10 Ko. Pour plus d'informations sur le balisage des ressources DynamoDB et les restrictions de balisage, consultez Étiquetage de ressources dans DynamoDB et the section called "Restrictions d'étiquetage dans DynamoDB".

Pour plus d'informations sur l'utilisation de balises pour contrôler l'accès aux AWS ressources, consultez les rubriques suivantes du guide de l'utilisateur IAM :

- À quoi sert ABAC AWS
- Contrôle de l'accès aux AWS ressources à l'aide de balises

À l'aide d'ABAC, vous pouvez appliquer différents niveaux d'accès à vos équipes et à vos applications afin qu'elles puissent effectuer des actions sur les tables DynamoDB en utilisant moins de politiques. Vous pouvez spécifier une balise dans le champ élément de condition d'une politique IAM pour contrôler l'accès à vos tables ou index DynamoDB. Ces conditions déterminent le niveau d'accès d'un principal, d'un utilisateur ou d'un rôle IAM aux tables et aux index DynamoDB. Lorsqu'un principal IAM fait une demande d'accès à DynamoDB, les balises de la ressource et de l'identité sont évaluées par rapport aux conditions des balises définies dans la politique IAM. Par la suite, la politique n'entre en vigueur que si les conditions relatives aux balises sont remplies. Cela vous permet de créer une politique IAM qui énonce efficacement l'un des éléments suivants :

- Autorisez l'utilisateur à gérer uniquement les ressources dotées d'une balise comportant une clé *X* et une valeur *Y*.
- Refusez l'accès à tous les utilisateurs pour les ressources étiquetées avec une clé *X*.

Par exemple, vous pouvez créer une politique qui permet aux utilisateurs de mettre à jour une table uniquement si elle possède la balise de paire clé-valeur : `"environment": "staging"`. Vous pouvez utiliser la clé de ResourceTag condition aws : pour autoriser ou refuser l'accès à une table en fonction des balises associées à cette table.

Vous pouvez inclure des conditions basées sur les attributs lors de la création de la politique ou ultérieurement à l'aide de l'AWS API AWS Management Console,AWS Command Line Interface(AWS CLI), du AWS SDK ou.AWS CloudFormation

L'exemple suivant autorise l'UpdateItemaction sur une table nommée `MusicTable` si elle inclut une clé de balise avec le nom `environment` et la valeur`production`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Rubriques

- Pourquoi utiliser l’ABAC ?
- Clés de condition pour mettre en œuvre ABAC avec DynamoDB
- Considérations relatives à l’utilisation d’ABAC avec DynamoDB
- Activation de l’ABAC dans DynamoDB
- Utilisation de l’ABAC avec des tables et des index DynamoDB

- Exemples d'utilisation de l'ABAC avec des tables et des index DynamoDB
- Résolution des erreurs courantes liées à l'ABAC pour les tables et les index DynamoDB

Pourquoi utiliser l'ABAC ?

- Gestion des politiques simplifiée : vous utilisez moins de politiques, car vous n'avez pas à créer de politiques différentes afin de définir le niveau d'accès pour chaque principal IAM.
- Contrôle d'accès pouvait être mis à l'échelle : la mise à l'échelle du contrôle d'accès est plus facile avec ABAC, car vous n'avez pas à mettre à jour vos politiques lorsque vous créez de nouvelles ressources DynamoDB. Vous pouvez utiliser des balises pour autoriser l'accès aux principaux IAM qui contiennent des balises correspondant à celles de la ressource. Vous pouvez intégrer de nouveaux principaux IAM ou de nouvelles ressources DynamoDB et appliquer les balises appropriées afin d'accorder automatiquement les autorisations nécessaires sans avoir à modifier les politiques.
- Gestion précise des autorisations : il est recommandé d'accorder le moindre privilège lorsque vous créez des politiques. Avec ABAC, vous pouvez créer des balises pour le principal IAM et les utiliser pour restreindre l'accès à des actions et à des ressources spécifiques correspondant aux balises du principal IAM.
- Alignement avec l'annuaire d'entreprise : vous pouvez mapper des balises avec les attributs des employés existants à partir de votre annuaire d'entreprise afin d'aligner vos politiques de contrôle d'accès sur votre structure organisationnelle.

Clés de condition pour mettre en œuvre ABAC avec DynamoDB

Vous pouvez utiliser les clés de condition suivantes dans vos AWS politiques pour contrôler le niveau d'accès à vos tables et index DynamoDB :

- aws : ResourceTag /tag-key : contrôle l'accès selon que la paire clé-valeur de balise d'une table ou d'un index DynamoDB correspond ou non à la clé et à la valeur de balise d'une politique. Cette clé de condition s'applique à tous APIs ceux qui opèrent sur une table ou un index existant.

  Les conditions `dynamodb:ResourceTag` sont évaluées comme si vous n'aviez attaché aucune balise à une ressource.
- aws : RequestTag /tag-key : permet de comparer la paire clé-valeur de balise transmise dans la demande avec la paire de balises que vous spécifiez dans la politique. Cette clé de condition

est pertinente pour ceux APIs qui contiennent des balises dans le cadre de la charge utile de la demande. Ceux-ci APIs incluent CreateTableet TagResource.

- aws : TagKeys : Compare les clés de balise d'une demande avec les clés que vous spécifiez dans la politique. Cette clé de condition est pertinente pour ceux APIs qui contiennent des balises dans le cadre de la charge utile de la demande. Ceux-ci APIs incluent `CreateTableTagResource`, et`UntagResource`.

Considérations relatives à l'utilisation d'ABAC avec DynamoDB

Lorsque vous utilisez ABAC avec des tables ou des index DynamoDB, les considérations suivantes s'appliquent :

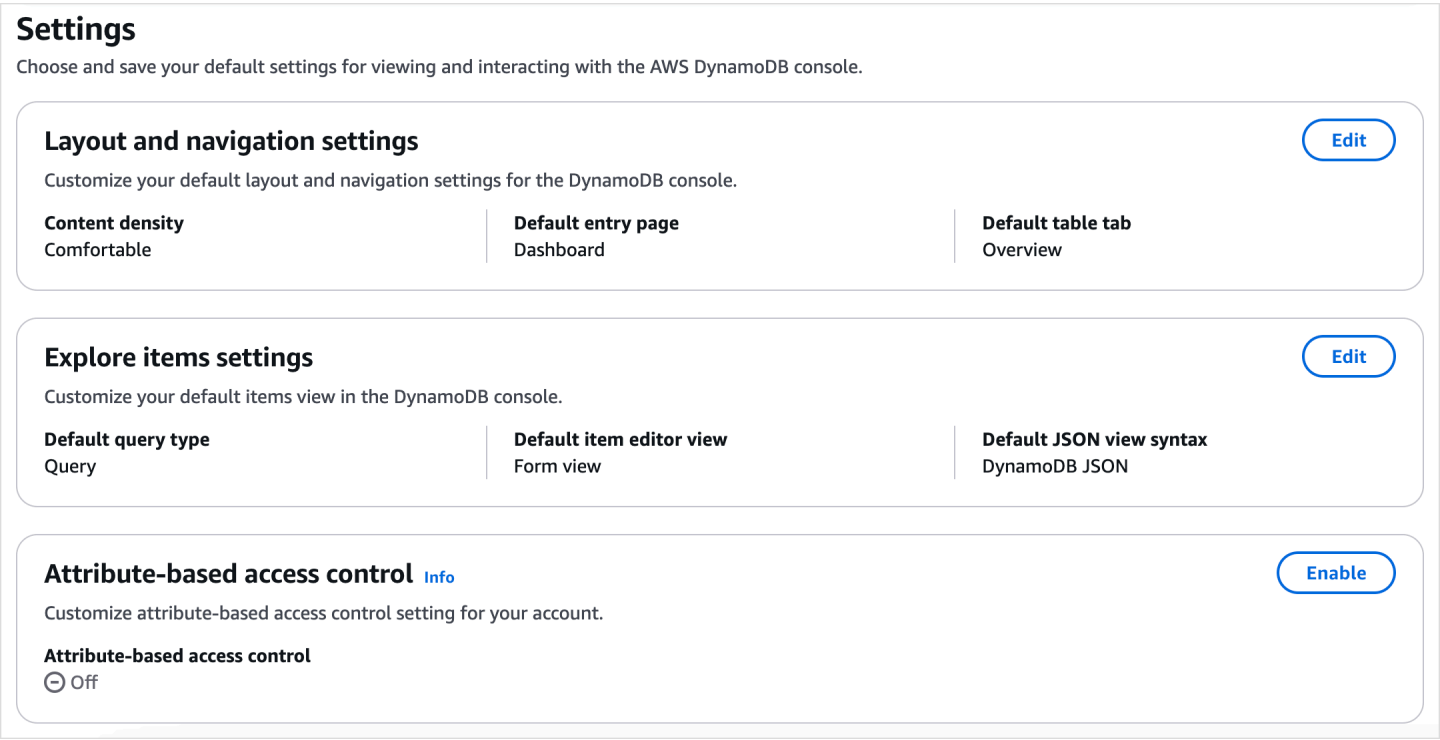
- Le balisage et l'ABAC ne sont pas pris en charge pour DynamoDB Streams.
- Le balisage et l'ABAC ne sont pas pris en charge pour les sauvegardes DynamoDB. Pour utiliser l'ABAC avec des sauvegardes, nous vous recommandons d'utiliser AWS Backup.
- Les balises ne sont pas conservées dans les tables restaurées. Vous devez ajouter des balises aux tables restaurées avant de pouvoir utiliser des conditions basées sur des balises dans vos politiques.

Activation de l'ABAC dans DynamoDB

Dans la plupart des cas Comptes AWS, ABAC est activé par défaut. À l'aide de la console DynamoDB, vous pouvez vérifier si ABAC est activé pour votre compte. Pour ce faire, assurez-vous d'ouvrir la console DynamoDB avec un rôle doté de l'autorisation dynamodb :. GetAbacStatus Ouvrez ensuite la page Paramètres de la console DynamoDB.

Si vous ne voyez pas la carte Contrôle d'accès basé sur les attributs ou si le statut de la carte est Activé, cela signifie que l'ABAC est activé pour votre compte. Toutefois, si vous voyez la carte de Contrôle d'accès basé sur les attributs avec le statut Désactivé, comme indiqué dans l'image suivante, cela signifie que l'ABAC n'est pas activé pour votre compte.

Contrôle d’accès par attributs - Non activé



L'ABAC n'est pas activéComptes AWS pour les conditions basées sur les balises spécifiées dans leurs politiques basées sur l'identité ou dans d'autres politiques qui doivent encore être auditées. Si l’ABAC n’est pas activé pour votre compte, les conditions basées sur les balises de vos politiques destinées à agir sur les tables ou les index DynamoDB sont évaluées comme si aucune balise n’était présente pour vos ressources ou vos demandes d’API. Lorsque l’ABAC est activé pour votre compte, les conditions basées sur les balises figurant dans les politiques de votre compte sont évaluées en tenant compte des balises associées à vos tables ou à vos demandes d’API.

Pour activer l’ABAC pour votre compte, nous vous recommandons de commencer par auditer vos politiques comme décrit dans la section Audit de politique. Incluez ensuite les autorisations requises pour l’ABAC dans votre politique IAM. Enfin, suivez les étapes décrites dans Activation de l’ABAC dans la console pour activer l’ABAC pour votre compte dans la région actuelle. Après avoir activé l’ABAC, vous pouvez le désactiver dans les sept prochains jours calendaires suivant l’activation.

Rubriques

- Audit de vos politiques avant d’activer l’ABAC
- Autorisations IAM requises pour activer l’ABAC
- Activation de l’ABAC dans la console

Audit de vos politiques avant d'activer l'ABAC

Avant d'activer l'ABAC pour votre compte, auditez vos politiques pour vous assurer que les conditions basées sur des balises qui peuvent exister dans les politiques de votre compte sont configurées comme prévu. L'audit de vos politiques permet d'éviter les surprises liées aux modifications d'autorisation apportées à vos flux de travail DynamoDB après l'activation de l'ABAC. Pour consulter des exemples d'utilisation de conditions basées sur les attributs avec des balises, ainsi que le comportement avant et après de la mise en œuvre de l'ABAC, consultez Exemples d'utilisation de l'ABAC avec des tables et des index DynamoDB.

Autorisations IAM requises pour activer l'ABAC

Vous devez disposer de l'autorisation `dynamodb:UpdateAbacStatus` pour activer l'ABAC pour votre compte dans la région actuelle. Pour vérifier si l'ABAC est activé pour votre compte, vous devez également avoir l'autorisation `dynamodb:GetAbacStatus`. Cette autorisation vous permet de consulter le statut de l'ABAC d'un compte dans n'importe quelle région. Vous avez besoin de ces autorisations en plus de l'autorisation nécessaire pour accéder à la console DynamoDB.

La politique IAM suivante accorde l'autorisation d'activer l'ABAC et de consulter son statut pour un compte dans la région actuelle.

```
{
"version": "2012-10-17",        &TCX5-2025-waiver;
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Activation de l'ABAC dans la console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/

2. Dans le panneau de navigation supérieur, sélectionnez la région pour laquelle vous souhaitez activer l'ABAC.
3. Dans le panneau de navigation de gauche, choisissez Paramètres.
4. Sur la page Settings (Paramètres), procédez comme suit :

    a. Dans la carte de Contrôle d'accès par attributs, choisissez Activer.

    b. Dans la case Confirmer le réglage du contrôle d'accès par attributs, choisissez Activer pour confirmer votre choix.

    Cette opération active l'ABAC pour la région actuelle et la carte de Contrôle d'accès basé sur les attributs indique le statut Activé.

    Si vous souhaitez désactiver l'ABAC après l'avoir activé sur votre console, vous pouvez le faire dans les sept jours calendaires suivant l'activation. Pour désactiver l'ABAC, choisissez Désactiver dans la carte de Contrôle d'accès par attributs de la page Paramètres.

    > **Note**
    >
    > La mise à jour du statut de l'ABAC est une opération asynchrone. Si les balises de vos politiques ne sont pas évaluées immédiatement, vous devrez peut-être attendre un certain temps, car l'application des modifications sera finalement cohérente.

Utilisation de l'ABAC avec des tables et des index DynamoDB

Les étapes suivantes montrent comment configurer des autorisations à l'aide de l'ABAC. Dans cet exemple de scénario, vous allez ajouter des balises à une table DynamoDB et créer un rôle IAM avec une politique qui inclut des conditions basées sur des balises. Vous allez ensuite tester les autorisations accordées dans la table DynamoDB en respectant les conditions des balises.

Rubriques

- Étape 1 : Ajouter des balises à une table DynamoDB
- Étape 2 : Créer un rôle IAM avec une politique incluant des conditions basées sur des balises
- Étape 3 : Tester les autorisations accordées

Étape 1 : Ajouter des balises à une table DynamoDB

Vous pouvez ajouter des balises à des tables DynamoDB nouvelles ou existantes à l'aide de l'API AWS,AWS CLI()AWS Command Line InterfaceAWS,AWS Management Console du SDK ou.AWS CloudFormation Par exemple, la commande CLI tag-resource suivante ajoute une balise à une table nommée `MusicTable`.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Étape 2 : Créer un rôle IAM avec une politique incluant des conditions basées sur des balises

Créez une politique IAM à l'aide de la clé de condition aws : ResourceTag /tag-key pour comparer la paire clé-valeur de balise spécifiée dans la politique IAM avec la paire clé-valeur attachée à la table. L'exemple de politique suivant permet aux utilisateurs de placer ou de mettre à jour des éléments dans des tables si celles-ci contiennent la paire clé-valeur de balise : `"environment": "staging"`. Si une table ne possède pas la paire clé-valeur de balise spécifiée, ces actions sont refusées.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Étape 3 : Tester les autorisations accordées

1. Attachez la politique IAM à un utilisateur ou à un rôle de test dans votre Compte AWS. Assurez-vous que le principal IAM que vous utilisez n'a pas déjà accès à la table DynamoDB par le biais d'une autre politique.
2. Assurez-vous que votre table DynamoDB contient la clé de balise `"environment"` avec une valeur de `"staging"`.
3. Exécutez les actions `dynamodb:PutItem` et `dynamodb:UpdateItem` et sur la table balisée. Ces actions devraient réussir si la paire clé-valeur de la balise `"environment": "staging"` est présente.

   Si vous effectuez ces actions sur une table dépourvue de la paire clé-valeur de balise `"environment": "staging"`, votre demande échouera avec un message d'erreur `AccessDeniedException`.

Vous pouvez également consulter les autres exemples de cas d'utilisation décrits dans la section suivante pour mettre en œuvre l'ABAC et effectuer d'autres tests.

Exemples d'utilisation de l'ABAC avec des tables et des index DynamoDB

Les exemples suivants illustrent certains cas d'utilisation pour la mise en œuvre des conditions basées sur les attributs à l'aide de balises.

Rubriques

- Exemple 1 : Autoriser une action à l'aide de aws : ResourceTag
- Exemple 2 : Autoriser une action à l'aide de aws : RequestTag
- Exemple 3 : Refuser une action à l'aide de aws : TagKeys

Exemple 1 : Autoriser une action à l'aide de aws : ResourceTag

À l'aide de la clé de condition `aws:ResourceTag/tag-key`, vous pouvez comparer la paire clé-valeur de balise spécifiée dans une politique IAM avec la paire clé-valeur attachée dans une table DynamoDB. Par exemple, vous pouvez autoriser une action spécifique, par exemple si les conditions des balises correspondent dans une politique IAM et dans une table. PutItem Effectuez les étapes suivantes pour ce faire :

Using the AWS CLI

1. Créer une table. L'exemple suivant utilise la AWS CLI commande create-table pour créer une table nommée. `myMusicTable`

```
aws dynamodb create-table \
  --table-name myMusicTable \
  --attribute-definitions AttributeName=id,AttributeType=S \
  --key-schema AttributeName=id,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --region us-east-1
```

2. Ajoutez une balise à cette table. L'exemple de AWS CLI commande tag-resource suivant ajoute la paire clé-valeur tag au. `Title: ProductManager` `myMusicTable`

```
aws dynamodb tag-resource --region us-east-1 --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/myMusicTable --tags Key=Title,Value=ProductManager
```

3. Créez une politique intégrée et ajoutez-la à un rôle auquel la politique AmazonDynamoDBReadOnlyAccessAWS gérée est attachée, comme indiqué dans l'exemple suivant.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Cette politique autorise l'action `PutItem` sur la table lorsque la clé de balise et la valeur associées à la table correspondent aux balises spécifiées dans la politique.

4. Assumez le rôle avec les politiques décrites à l'étape 3.
5. Utilisez la AWS CLI commande put-item pour placer un élément dans le. `myMusicTable`

```
aws dynamodb put-item \
    --table-name myMusicTable --region us-east-1 \
    --item '{
        "id": {"S": "2023"},
        "title": {"S": "Happy Day"},
        "info": {"M": {
            "rating": {"N": "9"},
            "Artists": {"L": [{"S": "Acme Band"}, {"S": "No One You Know"}]},
            "release_date": {"S": "2023-07-21"}
        }}
    }'
```

6. Analysez la table pour vérifier si l'élément y a été ajouté.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Créer une table. L'exemple suivant utilise l'CreateTableAPI pour créer une table nomméemyMusicTable.

```
DynamoDbClient dynamoDB = DynamoDbClient.builder().region(region).build();
CreateTableRequest createTableRequest = CreateTableRequest.builder()
    .attributeDefinitions(
        Arrays.asList(
            AttributeDefinition.builder()
            .attributeName("id")
            .attributeType(ScalarAttributeType.S)
            .build()
        )
    )
    .keySchema(
        Arrays.asList(
            KeySchemaElement.builder()
```

```
            .attributeName("id")
            .keyType(KeyType.HASH)
            .build()
        )
    )
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(5L)
        .writeCapacityUnits(5L)
        .build()
    )
    .tableName("myMusicTable")
    .build();

CreateTableResponse createTableResponse =
 dynamoDB.createTable(createTableRequest);
String tableArn = createTableResponse.tableDescription().tableArn();
String tableName = createTableResponse.tableDescription().tableName();
```

2. Ajoutez une balise à cette table. Dans l'exemple suivant, l'TagResourceAPI ajoute la paire clé-valeur de balise `Title: ProductManager` au. `myMusicTable`

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. Créez une politique intégrée et ajoutez-la à un rôle auquel la politique AmazonDynamoDBReadOnlyAccessAWS gérée est attachée, comme indiqué dans l'exemple suivant.

   JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Cette politique autorise l'action `PutItem` sur la table lorsque la clé de balise et la valeur associées à la table correspondent aux balises spécifiées dans la politique.

4. Assumez le rôle avec les politiques décrites à l'étape 3.
5. Utilisez l'PutItemAPI pour placer un élément dans le`myMusicTable`.

```
HashMap<String, AttributeValue> info = new HashMap<>();
info.put("rating", AttributeValue.builder().s("9").build());
info.put("artists", AttributeValue.builder().ss(List.of("Acme Band","No One You
 Know").build());
info.put("release_date", AttributeValue.builder().s("2023-07-21").build());

HashMap<String, AttributeValue> itemValues = new HashMap<>();
itemValues.put("id", AttributeValue.builder().s("2023").build());
itemValues.put("title", AttributeValue.builder().s("Happy Day").build());
itemValues.put("info", AttributeValue.builder().m(info).build());


PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
dynamoDB.putItem(putItemRequest);
```

6. Analysez la table pour vérifier si l'élément y a été ajouté.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
```

```
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK pour Python (Boto3)

1. Créer une table. L'exemple suivant utilise l'CreateTableAPI pour créer une table nomméemyMusicTable.

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    },
)

table_arn = create_table_response['TableDescription']['TableArn']
```

2. Ajoutez une balise à cette table. Dans l'exemple suivant, l'TagResourceAPI ajoute la paire clé-valeur de balise `Title: ProductManager` au. myMusicTable

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
```

```
)
```

3. Créez une politique intégrée et ajoutez-la à un rôle auquel la politique AmazonDynamoDBReadOnlyAccessAWS gérée est attachée, comme indiqué dans l'exemple suivant.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
           "Effect": "Allow",
           "Action": "dynamodb:PutItem",
           "Resource": "arn:aws:dynamodb:*:*:table/*",
           "Condition": {
               "StringEquals": {
               "aws:ResourceTag/Title": "ProductManager"
               }
           }
           }
       ]
       }
   ```

   Cette politique autorise l'action `PutItem` sur la table lorsque la clé de balise et la valeur associées à la table correspondent aux balises spécifiées dans la politique.

4. Assumez le rôle avec les politiques décrites à l'étape 3.
5. Utilisez l'PutItemAPI pour placer un élément dans le`myMusicTable`.

   ```
   put_item_response = client.put_item(
       TableName = 'myMusicTable'
       Item = {
           'id': '2023',
           'title': 'Happy Day',
           'info': {
               'rating': '9',
               'artists': ['Acme Band','No One You Know'],
               'release_date': '2023-07-21'
           }
       }
   ```

```
)
```

6. Analysez la table pour vérifier si l'élément y a été ajouté.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Sans l'ABAC

Si ABAC n'est pas activé pour vous Compte AWS, les conditions de balise de la politique IAM et de la table DynamoDB ne correspondent pas. Par conséquent, l'action `PutItem` renvoie le code `AccessDeniedException` en raison de l'effet de la politique `AmazonDynamoDBReadOnlyAccess`.

```
An error occurred (AccessDeniedException) when calling the PutItem operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoDBReadOnlyAccess/Alice is
 not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable because no identity-based policy allows the
 dynamodb:PutItem action.
```

Avec l'ABAC

Si ABAC est activé pour vous Compte AWS, l'`put-item`action se termine correctement et ajoute un nouvel élément à votre tableau. Cela est dû au fait que la politique intégrée de la table autorise l'action `PutItem` si les conditions de balise de la politique IAM et de la table correspondent.

Exemple 2 : Autoriser une action à l'aide de aws : RequestTag

À l'aide de la RequestTagclé de condition aws : /tag-key, vous pouvez comparer la paire clé-valeur de balise transmise dans votre demande avec la paire de balises spécifiée dans la politique IAM. Vous pouvez autoriser une action spécifique, telle que `CreateTable`, en utilisant `aws:RequestTag` si les conditions de la balise ne correspondent pas. Effectuez les étapes suivantes pour ce faire :

Using the AWS CLI

1. Créez une politique intégrée et ajoutez-la à un rôle auquel la politique AmazonDynamoDBReadOnlyAccessAWS gérée est attachée, comme indiqué dans l'exemple suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Créez une table contenant la paire clé-valeur de balise de `"Owner": "John"`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Owner,Value=John \
--table-name myMusicTable
```

Using the AWS SDK pour Python (Boto3)

1. Créez une politique intégrée et ajoutez-la à un rôle auquel la politique AmazonDynamoDBReadOnlyAccessAWS gérée est attachée, comme indiqué dans l'exemple suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Créez une table contenant la paire clé-valeur de balise de `"Owner": "John"`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
```

```
    },
    Tags=[
        {
            'Key': 'Owner',
            'Value': 'John'
        },
    ],
)
```

Sans l'ABAC

Si ABAC n'est pas activé pour vous Compte AWS, les conditions de balise de la politique intégrée et de la table DynamoDB ne correspondent pas. Par conséquent, la demande `CreateTable` échoue et votre table n'est pas créée.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/Admin/John is not authorized to perform:
 dynamodb:CreateTable on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
myMusicTable because no identity-based policy allows the dynamodb:CreateTable action.
```

Avec l'ABAC

Si ABAC est activé pour vous Compte AWS, votre demande de création de table est terminée avec succès. Étant donné que la paire clé-valeur de la balise `"Owner": "John"` est présente dans la demande `CreateTable`, la politique en ligne permet à l'utilisateur `John` d'effectuer l'action `CreateTable`.

Exemple 3 : Refuser une action à l'aide de aws : TagKeys

À l'aide de la clé de TagKeys condition aws :, vous pouvez comparer les clés de balise d'une demande avec les clés spécifiées dans la politique IAM. Vous pouvez refuser une action spécifique, par exemple `CreateTable`, avec `aws:TagKeys` si une clé de balise spécifique n'est pas présente dans la demande. Effectuez les étapes suivantes pour ce faire :

Using the AWS CLI

1. Ajoutez une politique gérée par le client à un rôle auquel est attachée la politique AWS gérée d'AmazonDynamoDBFullaccès, comme indiqué dans l'exemple suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Assumez le rôle auquel la politique était attachée et créez une table avec la clé de balise `Title`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Title,Value=ProductManager \
--table-name myMusicTable
```

Using the AWS SDK pour Python (Boto3)

1. Ajoutez une politique gérée par le client à un rôle auquel est attachée la politique AWS gérée d'AmazonDynamoDBFullaccès, comme indiqué dans l'exemple suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Assumez le rôle auquel la politique était attachée et créez une table avec la clé de balise `Title`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
```

```
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ],
)
```

Sans l'ABAC

Si ABAC n'est pas activé pour vous Compte AWS, DynamoDB n'envoie pas les clés de balise de la commande à `create-table` IAM. La condition `Null` garantit que la condition a la valeur `false` s'il n'y a pas de clés de balise dans la demande. Comme la politique `Deny` ne correspond pas, la commande `create-table` s'exécute correctement.

Avec l'ABAC

Si ABAC est activé pour vous Compte AWS, les clés de balise transmises dans la `create-table` commande sont transmises à IAM. La clé de balise `Title` est évaluée par rapport à la clé de balise basée sur les conditions, `CostCenter`, présente dans la politique `Deny`. La clé de balise `Title` ne correspond pas à la clé de balise présente dans la politique `Deny` à cause de l'opérateur `StringNotEquals`. Par conséquent, l'action `CreateTable` échoue et votre table n'est pas créée. L'exécution de la commande `create-table` renvoie le message d'erreur `AccessDeniedException`.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoFullAccessRole/ProductManager is
 not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable with an explicit deny in an identity-based
 policy.
```

Résolution des erreurs courantes liées à l'ABAC pour les tables et les index DynamoDB

Cette rubrique fournit des conseils de résolution pour les erreurs et problèmes courants que vous pouvez rencontrer lorsque vous mettez en œuvre l'ABAC dans des tables ou des index DynamoDB.

Les clés de condition spécifiques au service dans les politiques génèrent une erreur

Les clés de condition spécifiques au service ne sont pas considérées comme des clés de condition valides. Si vous avez utilisé ce type de clés dans vos politiques, cela provoquera une erreur. Pour résoudre ce problème, vous devez remplacer les clés de condition spécifiques au service par une clé de condition appropriée pour mettre en œuvre l'ABAC dans DynamoDB.

Supposons, par exemple, que vous ayez utilisé la clé de `dynamodb:ResourceTag` condition dans une politique intégrée qui exécute la PutItemdemande. Imaginez que la demande échoue avec le message d'erreur `AccessDeniedException`. L'exemple suivant montre la politique en ligne erronée avec la clé de condition `dynamodb:ResourceTag`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Pour résoudre ce problème, remplacez la clé de condition `dynamodb:ResourceTag` par `aws:ResourceTag`, comme indiqué dans l'exemple suivant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Impossible de désactiver l'ABAC

Si ABAC a été activé pour votre compte via Support, vous ne pourrez pas désactiver ABAC via la console DynamoDB. Pour désactiver l'ABAC, contactez Support.

Vous ne pouvez désactiver vous-même l'ABAC que si les conditions suivantes sont réunies :

- Vous avez utilisé la méthode en libre-service pour l'activer via la console DynamoDB.
- Vous le désactiver dans les sept jours civils suivant son activation.

Protection des données dans DynamoDB

Amazon DynamoDB offre une infrastructure de stockage hautement durable, conçue pour un stockage de données principal et stratégique. Les données sont stockées de façon redondante sur plusieurs appareils situés dans plusieurs installations au sein d'une région Amazon DynamoDB.

DynamoDB protège les données utilisateur stockées au repos ainsi que les données en transit entre les clients locaux et DynamoDB, ainsi qu'entre DynamoDB et d'autres ressources au sein de la même région.AWSAWS

Rubriques

- Chiffrement de DynamoDB au repos
- Sécurisation des connexions DynamoDB à l’aide de points de terminaison d’un VPC et de politiques IAM

Chiffrement de DynamoDB au repos

Toutes les données utilisateur stockées dans Amazon DynamoDB sont entièrement chiffrées au repos. Le chiffrement au repos de DynamoDB offre une sécurité renforcée en chiffrant toutes vos données au repos à l’aide de clés de chiffrement stockées dans AWS Key Management Service(AWS KMS). Cette fonctionnalité réduit la lourdeur opérationnelle et la complexité induites par la protection des données sensibles. Le chiffrement au repos vous permet de créer des applications sensibles en matière de sécurité qui sont conformes aux exigences réglementaires et de chiffrement strictes.

Le chiffrement au repos de DynamoDB offre une couche supplémentaire de protection des données en les sécurisant toujours dans une table chiffrée incluant une clé primaire, des index secondaires locaux et globaux, des flux, des tables globales, des sauvegardes et des clusters DynamoDB Accelerator (DAX) chaque fois que les données sont stockées sur un support durable. Les politiques organisationnelles et les réglementations sectorielles ou gouvernementales, ainsi que les exigences de conformité, exigent souvent l’utilisation du chiffrement au repos pour augmenter la sécurité des données de vos applications. Pour plus d’informations sur le chiffrement des applications de base de données, consultez AWS Database Encryption SDK.

Encryption at rest s'intègre AWS KMSà la gestion des clés de chiffrement utilisées pour chiffrer vos tables. Pour plus d’informations sur les types de clés et leurs états, consultez Concepts AWS Key Management Service dans le Guide du développeur AWS Key Management Service.

Lorsque vous créez une nouvelle table, vous pouvez choisir l'un des AWS KMS key types suivants pour chiffrer votre table. Vous pouvez passer d’un type de clé à l’autre à tout moment.

- Clé détenue par AWS— Type de cryptage par défaut. La clé est détenue par DynamoDB (sans frais supplémentaires).

- Clé gérée par AWS— La clé est enregistrée dans votre compte et est gérée par AWS KMS(des AWS KMS frais s'appliquent).
- Clé gérée par le client – La clé et stockée dans votre compte. Elle est créée, détenue et gérée par vous. Vous avez le contrôle total de la clé KMS (AWS KMS des frais s'appliquent).

Pour plus d'informations sur les types de clés, voir Clés et AWS clés clients.

> **Note**
>
> - Lors de la création d'un cluster DAX avec le chiffrement au repos activé, une cléClé gérée par AWS est utilisée pour chiffrer les données au repos dans le cluster.
> - Si votre table a une clé de tri, certaines clés de tri qui marquent les limites de plage sont stockées en texte brut dans les métadonnées de la table.

Lorsque vous accédez à une table chiffrée, DynamoDB déchiffre les données de la table de manière transparente. Vous n'avez pas besoin de changer de code ou d'application pour utiliser ou gérer des tables chiffrées. DynamoDB continue d'offrir la même latence inférieure à 10 millisecondes à laquelle vous êtes habitué, et toutes les interrogations de DynamoDB fonctionnent sans heurt sur vos données chiffrées.

Vous pouvez spécifier une clé de chiffrement lorsque vous créez une nouvelle table ou que vous changez les clés de chiffrement sur une table existante à l'aide de l'AWS Command Line Interface API AWS Management Console, (AWS CLI) ou Amazon DynamoDB. Pour savoir comment procéder, consultez Gestion des tables chiffrées dans DynamoDB.

Le chiffrement au repos à l'aide du Clé détenue par AWS est proposé sans frais supplémentaires. Cependant,AWS KMS des frais s'appliquent pour une clé gérée par le client Clé gérée par AWS et pour une telle clé. Pour plus d'informations sur la tarification, consultez Tarification d'AWS KMS.

Le chiffrement DynamoDB au repos est disponible dans AWS toutes les régions, y compris les régions de AWS Chine (Pékin) et de AWS Chine (Ningxia) et AWS GovCloud les régions (États-Unis). Pour plus d'informations, consultez Fonctionnement du chiffrement DynamoDB au repos et Notes d'utilisation du chiffrement de DynamoDB au repos.

Fonctionnement du chiffrement DynamoDB au repos

Le chiffrement Amazon DynamoDB au repos chiffre vos données à l'aide du chiffrement Advanced Encryption Standard 256 bits (AES-256), ce qui contribue à sécuriser vos données contre tout accès non autorisé au stockage sous-jacent.

Encryption at rest s'intègre àAWS Key Management Service(AWS KMS) pour gérer les clés de chiffrement utilisées pour chiffrer vos tables.

> **Note**
>
> En mai 2022, le calendrier de rotation AWS KMS a été modifié,Clés gérées par AWS passant de tous les trois ans (environ 1 095 jours) à chaque année (environ 365 jours).
> Clés gérées par AWS Les nouvelles versions font l'objet d'une rotation automatique un an après leur création, et environ chaque année par la suite.
> Clés gérées par AWS Les versions existantes font automatiquement l'objet d'une rotation un an après leur dernière rotation, puis chaque année.

Clés détenues par AWS

Clés détenues par AWS ne sont pas enregistrés dans votre AWS compte. Elles font partie d'un ensemble de clés KMS qui AWS possède et gère pour une utilisation dans plusieurs AWS comptes.AWS services que vous pouvez Clés détenues par AWS utiliser pour protéger vos données.Clés détenues par AWS utilisés par DynamoDB font l'objet d'une rotation annuelle (environ 365 jours).

Vous ne pouvez pas consulter, gérer Clés détenues par AWS, utiliser ou auditer leur utilisation. Toutefois, vous n'avez pas besoin de faire quelque travail que ce soit ou de modifier les programmes pour protéger les clés qui chiffrent vos données.

Aucuns frais mensuels ni frais d'utilisation ne vous sont facturés pour l'utilisation de Clés détenues par AWS, et ils ne sont pas pris en compte dans les AWS KMS quotas de votre compte.

Clés gérées par AWS

Clés gérées par AWS sont des clés KMS de votre compte créées, gérées et utilisées en votre nom par un AWS service intégré àAWS KMS. Vous pouvez afficher les clés Clés gérées par AWS dans votre compte, afficher leurs politiques de clé et contrôler leur utilisation dans les journaux AWS CloudTrail. Toutefois, vous ne pouvez pas gérer ces clés KMS ou modifier leurs autorisations.

Encryption at rest s'intègre automatiquement AWS KMSà DynamoDB `aws/dynamodb` () qui est utiliséClés gérées par AWS pour chiffrer vos tables pour gérer. S'il Clé gérée par AWS n'en existe pas lorsque vous créez votre table DynamoDB chiffrée AWS KMS, une nouvelle clé est automatiquement créée pour vous. Cette clé est utilisée avec les tables chiffrées créées dans le futur.AWS KMS combine du matériel et des logiciels sécurisés et hautement disponibles pour fournir un système de gestion des clés adapté au cloud.

Pour plus d'informations sur la gestion des autorisations du Clé gérée par AWS, voir Autoriser l'utilisation du Clé gérée par AWS dans le Guide du AWS Key Management Service développeur.

Clés gérées par le client

Les clés gérées par le client sont des clés KMS de votre AWS compte que vous créez, détenez et gérez. Vous disposez d'un contrôle total sur ces clés KMS, incluant l'établissement et la gestion de leurs politiques de clé, politiques IAM et octrois, leur activation et désactivation, la rotation de leur contenu de chiffrement, l'ajout d'étiquettes, la création d'alias qui y font référence, et la planification de leur suppression. Pour plus d'informations sur la gestion des autorisations d'une clé gérée par le client, consultez Clés gérées par le client.

Lorsque vous spécifiez une clé gérée par le client comme clé de chiffrement au niveau de la table, la table DynamoDB, les index secondaires locaux et globaux, ainsi que les flux sont chiffrés avec la même clé. Les sauvegardes à la demande sont chiffrées à l'aide de la clé de chiffrement au niveau de la table spécifiée au moment de leur création. La mise à jour de la clé de chiffrement au niveau de la table n'a pas pour effet de modifier la clé de chiffrement associée aux sauvegardes à la demande existantes.

La désactivation de la clé gérée par le client ou la planification de sa suppression empêchent tous les utilisateurs et le service DynamoDB de chiffrer ou déchiffrer des données, ainsi que d'effectuer des opérations de lecture et d'écriture sur la table. Pour vous permettre de continuer à accéder à votre table et vous éviter de perdre des données, DynamoDB doit avoir accès à votre clé de chiffrement.

Si vous désactivez votre clé gérée par le client ou planifiez sa suppression, l'état de votre table devient Inaccessible. Pour vous assurer de pouvoir continuer à utiliser la table, vous devez fournir à DynamoDB l'accès à la clé de chiffrement spécifiée dans un délai de sept jours. Dès que le service détecte que votre clé de chiffrement est inaccessible, DynamoDB vous envoie une notification par e-mail pour vous alerter.

> **Note**
>
> - Si votre clé gérée par le client reste inaccessible au service DynamoDB pendant plus de sept jours, la table est archivée et n'est plus accessible. DynamoDB crée une sauvegarde à la demande de votre table et vous êtes facturé pour cela. Vous pouvez utiliser cette sauvegarde à la demande pour restaurer vos données vers une nouvelle table. Pour lancer la restauration, la dernière clé gérée par le client sur la table doit être activée et DynamoDB doit y avoir accès.
> - Si la clé gérée par le client qui a été utilisée pour chiffrer un réplica de table globale est inaccessible, DynamoDB supprime ce réplica du groupe de réplication. Le réplica n'est pas supprimé et la réplication depuis et vers cette région s'arrête 20 heures après que l'inaccessibilité de la clé gérée par le client a été détectée.

Pour plus d'informations, consultez les sections Activation des clés et Suppression de clés.

Remarques sur l'utilisation Clés gérées par AWS

Amazon DynamoDB ne peut pas lire les données de votre table s'il n'a pas accès à la clé KMS enregistrée dans votre compte.AWS KMS DynamoDB utilise un chiffrement d'enveloppe et une hiérarchie de clés pour chiffrer les données. Votre clé de AWS KMS chiffrement est utilisée pour chiffrer la clé racine de cette hiérarchie de clés. Pour plus d'informations, consultez Chiffrement d'enveloppe dans le Guide du développeur AWS Key Management Service.

DynamoDB n'appelle pas AWS KMS toutes les opérations DynamoDB. La clé est actualisée une fois toutes les 5 minutes pour chaque appelant ayant un trafic actif.

Assurez-vous d'avoir configuré le kit SDK pour réutiliser les connexions. Dans le cas contraire, DynamoDB devra rétablir de nouvelles entrées de AWS KMS cache pour chaque opération DynamoDB avec des latences. En outre, vous pourriez avoir à faire face àAWS KMS des CloudTrail coûts plus élevés. Par exemple, pour faire cela à l'aide du kit SDK Node.js, vous pouvez créer un agent HTTPS avec le paramètre `keepAlive` activé. Pour plus d'informations, consultez Configuration de keepAlive dans Node.js dans le Guide du développeur AWS SDK pour JavaScript.

Notes d'utilisation du chiffrement de DynamoDB au repos

Lorsque vous utilisez un chiffrement au repos dans Amazon DynamoDB, tenez compte des considérations suivantes.

Toutes les données des tables sont chiffrées

Le chiffrement au repos côté serveur est activé sur toutes les données de tables DynamoDB et ne peut pas être désactivé. Vous ne pouvez pas chiffrer uniquement un sous-ensemble d'éléments dans une table.

La fonction de chiffrement au repos chiffre uniquement les données lorsque qu'elles sont statiques (au repos) sur un support de stockage permanent. Si la sécurité des données est un élément important pour les données en transit ou en cours d'utilisation, vous devrez peut-être prendre des mesures supplémentaires :

- Données en transit : toutes vos données dans DynamoDB sont chiffrées en transit. Par défaut, les communications avec DynamoDB utilisent le protocole HTTPS qui protège le trafic réseau à l'aide du chiffrement Secure Sockets Layer (SSL)/Transport Layer Security (TLS).
- Données en cours d'utilisation : protégez vos données avant de les envoyer à DynamoDB à l'aide d'un chiffrement côté client. Pour plus d'informations, consultez Chiffrement côté client et côté serveur dans le Guide du développeur de client de chiffrement Amazon DynamoDB.

Vous pouvez utiliser des flux avec des tables chiffrées. Les flux de DynamoDB sont toujours chiffrés avec une clé de chiffrement au niveau de la table. Pour de plus amples informations, veuillez consulter Modifier la récupération de données pour DynamoDB Streams.

Les sauvegardes de DynamoDB sont chiffrées, et le chiffrement est également activé dans les tables restaurées à partir de ces sauvegardes. Vous pouvez utiliser la cléClé détenue par AWSClé gérée par AWS, ou la clé gérée par le client pour chiffrer vos données de sauvegarde. Pour de plus amples informations, veuillez consulter Sauvegarde et restauration pour DynamoDB.

Les index secondaires locaux et les index secondaires globaux sont chiffrés à l'aide de la même clé que la table de base.

Types de chiffrement

> **Note**
>
> Les clés gérées par le client ne sont pas prises en charge dans Global Table Version 2017. Si vous souhaitez utiliser une clé de ce type dans une table globale DynamoDB, vous devez mettre à niveau cette table vers Global Table Version 2019, puis l'activer.

Sur le AWS Management Console, le type de chiffrement correspond à l'utilisation `KMS` de la clé gérée par le client Clé gérée par AWS ou de la clé gérée par le client pour chiffrer vos données. Le type de chiffrement est `DEFAULT` lorsque vous utilisez la cléClé détenue par AWS. Dans l'API Amazon DynamoDB, le type de chiffrement correspond à l'utilisation de la clé gérée par le `KMS` client ou de la clé gérée Clé gérée par AWS par le client. Si le type de chiffrement n'est pas spécifié, vos données sont chiffrées avec la cléClé détenue par AWS. Vous pouvez basculer entre la cléClé détenue par AWSClé gérée par AWS, et la clé gérée par le client à tout moment. Vous pouvez utiliser la console, le AWS Command Line Interface(AWS CLI) ou l'API Amazon DynamoDB pour changer les clés de chiffrement.

Notez que les limites suivantes s'appliquent lors de l'utilisation des clés gérées par le client :

- Vous ne pouvez pas utiliser une clé gérée par le client avec des clusters DynamoDB Accelerator (DAX). Pour de plus amples informations, veuillez consulter Chiffrement au repos DAX.
- Vous pouvez utiliser une clé gérée par le client pour chiffrer les tables qui utilisent des transactions. Cependant, pour assurer la durabilité de la propagation des transactions, une copie de la demande de transaction est temporairement stockée par le service et chiffrée à l'aide d'une cléClé détenue par AWS. Les données validées dans vos tables et index secondaires sont toujours chiffrées au repos à l'aide de votre clé gérée par le client.
- Vous pouvez utiliser une clé gérée par le client pour chiffrer les tables qui utilisent Contributor Insights. Cependant, les données transmises àAmazon CloudWatch sont cryptées avec un Clé détenue par AWS.
- Lorsque vous passez à une nouvelle clé gérée par le client, veillez à conserver la clé d'origine activée jusqu'à ce que le processus soit terminé.AWS aura toujours besoin de la clé d'origine pour déchiffrer les données avant de les chiffrer avec la nouvelle clé. Le processus sera terminé lorsque le SSEDescription statut de la table sera ACTIVÉ et que KMSMaster KeyArn la nouvelle clé gérée par le client sera affichée. À ce stade, la clé d'origine peut être désactivée ou planifiée en vue de sa suppression.
- Une fois la nouvelle clé gérée par le client affichée, la table et toutes les nouvelles sauvegardes à la demande sont chiffrées avec la nouvelle clé.
- Toutes les sauvegardes à la demande existantes restent chiffrées avec la clé gérée par le client utilisée lors de la création de ces sauvegardes. Vous aurez besoin de la même clé pour restaurer ces sauvegardes. Vous pouvez identifier la clé pour la période au cours de laquelle chaque sauvegarde a été créée en utilisant l' DescribeBackup API pour afficher celles de cette sauvegarde SSEDescription.

- Si vous désactivez votre clé gérée par le client ou planifiez sa suppression, toutes les données dans DynamoDB Streams restent sujettes à une durée de vie de 24 heures. Toutes les données d'activité non récupérées sont éligibles à la suppression lorsqu'elles ont plus de 24 heures.
- Si vous désactivez votre clé gérée par le client ou planifiez sa suppression, les suppressions liées au Time-to-live (TTL) continuent pendant 30 minutes. Ces suppressions TTL continuent d'être émises vers DynamoDB Streams et sont soumises à l'intervalle standard. trimming/retention

  Pour plus d'informations, consultez les sections Activation des clés et Suppression de clés.

Utilisation des clés KMS et des clés de données

La fonctionnalité de chiffrement DynamoDB au repos utilise AWS KMS key une et une hiérarchie de clés de données pour protéger les données de votre table. DynamoDB utilise la même hiérarchie de clés pour protéger les DynamoDB Streams, les tables globales et les sauvegardes lorsqu'ils sont écrits sur un support durable.

Nous vous recommandons de planifier votre stratégie de chiffrement avant d'implémenter votre table dans DynamoDB. Si vous stockez des données sensibles ou confidentielles dans DynamoDB, pensez à inclure le chiffrement côté client dans votre stratégie. Vous pourrez ainsi chiffrer les données au plus près de leur origine et garantir leur protection tout au long de leur cycle de vie. Pour en savoir plus, consultez la documentation sur le client de chiffrement DynamoDB.

AWS KMS key

Le chiffrement au repos protège vos tables DynamoDB sous une AWS KMS key. Par défaut, DynamoDB utilise une Clé détenue par AWS, c'est-à-dire une clé de chiffrement à locataires multiples créée et gérée dans un compte de service DynamoDB. Toutefois, vous pouvez chiffrer vos tables DynamoDB sous une clé gérée par le client pour DynamoDB (`aws/dynamodb`) dans votre Compte AWS. Vous pouvez sélectionner une autre clé KMS pour chaque table. La clé KMS que vous sélectionnez pour une table est également utilisée pour chiffrer ses index secondaires, flux et sauvegardes locaux et globaux.

Vous sélectionnez la clé KMS pour une table lorsque vous créez ou mettez à jour la table. Vous pouvez modifier la clé KMS d'une table à tout moment, soit dans la console DynamoDB, soit en utilisant l'opération. UpdateTable Le processus de commutation des clés est transparent et ne provoque pas d'interruption ou de dégradation du service.

> **Important**
>
> DynamoDB ne prend en charge que les clés KMS symétriques. Vous ne pouvez pas utiliser une clé KMS asymétrique pour chiffrer vos tables DynamoDB.

Utilisez une clé gérée par le client pour obtenir les fonctions suivantes.

- Vous créez et gérez la clé KMS, y compris en définissant les politiques de clé, politiques IAM et octrois pour contrôler l'accès à la clé KMS. Vous pouvez activer et désactiver la clé KMS, activer et désactiver la rotation automatique des clés et supprimer la clé KMS lorsqu'elle n'est plus utilisée.
- Vous pouvez utiliser une clé gérée par le client avec un élément de clé importé ou dans un magasin de clés personnalisé que vous possédez et gérez.
- Vous pouvez vérifier le chiffrement et le déchiffrement de votre table DynamoDB en examinant les appels d'API DynamoDB dans les journaux.AWS KMSAWS CloudTrail

Utilisez le Clé gérée par AWS si vous avez besoin de l'une des fonctionnalités suivantes :

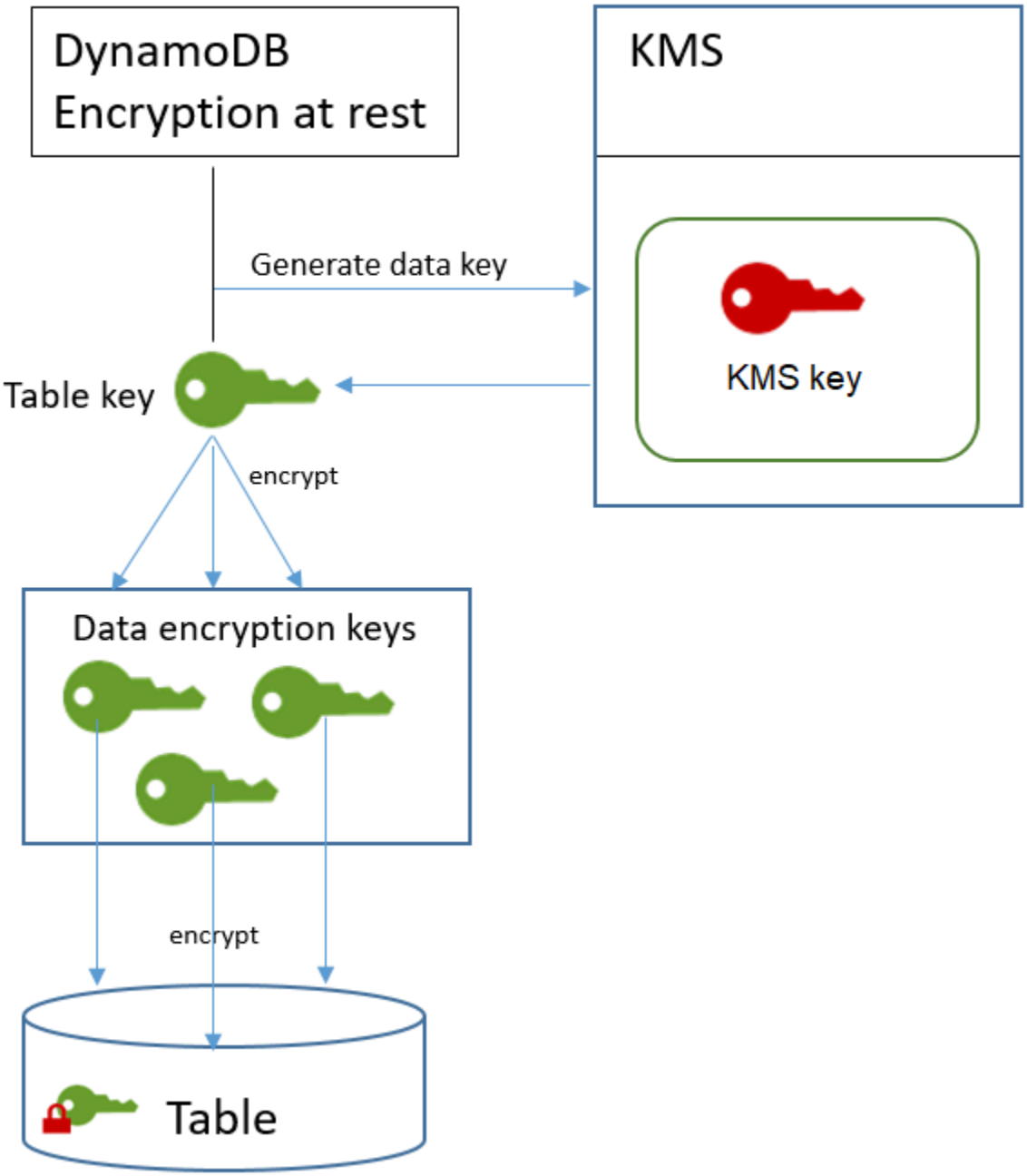
- Vous pouvez afficher la clé KMS et sa politique associée. (Vous ne pouvez pas modifier la politique de clé.)
- Vous pouvez vérifier le chiffrement et le déchiffrement de votre table DynamoDB en examinant les appels d'API DynamoDB dans les journaux.AWS KMSAWS CloudTrail

Cependant, il Clé détenue par AWS est gratuit et son utilisation n'est pas prise en compte dans les quotas de AWS KMS ressources ou de demandes. Les clés sont gérées par le client et Clés gérées par AWSdes frais sont facturés pour chaque appel d'API.AWS KMS Des quotas s'appliquent à ces clés KMS.

Clés de table

DynamoDB utilise la clé KMS de la table pour générer et chiffrer une clé de données unique pour la table, appelée clé de table. La clé de table est conservée pendant toute la durée de vie de la table chiffrée.

La clé de table est utilisée en tant que clé de chiffrement de clé. DynamoDB utilise cette clé de table pour protéger les clés de chiffrement des données utilisées pour chiffrer les données de la table. DynamoDB génère une clé de chiffrement des données unique pour chaque structure sous-jacente dans une table, mais plusieurs éléments de table peuvent être protégés par la même clé de chiffrement des données.



Lorsque vous accédez pour la première fois à une table chiffrée, DynamoDB envoie une demande d'utilisation de la clé KMS AWS KMS pour déchiffrer la clé de table. Ensuite, il utilise la clé de table en texte brut pour déchiffrer les clés de chiffrement de données et utilise les clés de chiffrement de données en texte brut pour déchiffrer les données de table.

DynamoDB stocke et utilise la clé de table et les clés de chiffrement des données en dehors de.AWS KMS Il protège toutes les clés avec les clés de chiffrement Advanced Encryption

Standard (AES) et 256 bits. Ensuite, il stocke les clés chiffrées avec les données chiffrées afin qu'elles soient disponibles pour déchiffrer les données de table à la demande.

Si vous modifiez la clé KMS de votre table, DynamoDB génère une nouvelle clé de table. Il utilise ensuite la nouvelle clé de table pour chiffrer de nouveau les clés de chiffrement des données.

Mise en cache des clés de table

Pour éviter d'appeler AWS KMS chaque opération DynamoDB, DynamoDB met en cache les clés de table en texte brut pour chaque appelant en mémoire. Si DynamoDB reçoit une demande pour la clé de table mise en cache après cinq minutes d'inactivité, il envoie une nouvelle demande AWS KMS pour déchiffrer la clé de table. Cet appel capturera toutes les modifications apportées aux politiques d'accès de la clé KMS dans AWS KMS ou Gestion des identités et des accès AWS(IAM) depuis la dernière demande de déchiffrement de la clé de table.

Autoriser l'utilisation de votre clé KMS

Si vous utilisez une clé gérée par le client ou la Clé gérée par AWS dans votre compte pour protéger votre table DynamoDB, les politiques associées à cette clé KMS doivent accorder à DynamoDB l'autorisation de l'utiliser en votre nom. Le contexte d'autorisation de DynamoDB inclut sa politique clé et octroie à cette dernière les autorisations nécessaires pour l'utiliser.Clé gérée par AWS

Vous avez un contrôle total des politiques et des octrois sur une clé gérée par le client. La Clé gérée par AWS est dans votre compte, vous pouvez donc consulter ses politiques et octrois. Mais, comme il est géré par AWS, vous ne pouvez pas modifier les politiques.

DynamoDB n'a pas besoin d'autorisation supplémentaire pour utiliser la Clé détenue par AWSvaleur par défaut afin de protéger les tables DynamoDB de votre.Compte AWS

Rubriques

- Politique clé pour un Clé gérée par AWS
- Politique de clé pour une clé gérée par le client
- Utilisation d'octrois pour autoriser DynamoDB

Politique clé pour un Clé gérée par AWS

Quand DynamoDB utilise la Clé gérée par AWS pour DynamoDB (`aws/dynamodb`) dans les opérations cryptographiques, il le fait au nom de l'utilisateur qui accède à la ressource DynamoDB.

La politique clé sur le Clé gérée par AWS donne à tous les utilisateurs du compte l'autorisation d'utiliser le Clé gérée par AWS pour des opérations spécifiées. Toutefois, l'autorisation est accordée uniquement lorsque DynamoDB effectue cette demande pour le compte de l'utilisateur. La ViaService condition énoncée dans la politique des clés n'autorise aucun utilisateur à utiliser le,Clé gérée par AWS sauf si la demande provient du service DynamoDB.

Cette politique clé, comme les politiques de toutes Clés gérées par AWS, est établie par AWS. Vous ne pouvez pas la modifier, mais vous pouvez la visualiser à tout moment. Pour plus d'informations, consultez Affichage d'une stratégie de clé.

Les instructions de politique de la politique de clé ont l'effet suivant :

- Autorisez les utilisateurs du compte à utiliser le Clé gérée par AWS for DynamoDB dans les opérations cryptographiques lorsque la demande provient de DynamoDB en leur nom. La politique autorise également les utilisateurs à créer des octrois pour la clé KMS.
- Autorise les identités IAM autorisées dans le compte à afficher les propriétés du Clé gérée par AWS pour DynamoDB et à révoquer l'octroi qui permet à DynamoDB d'utiliser la clé KMS. DynamoDB utilise des octrois pour les opérations de maintenance en cours.
- Permet à DynamoDB d'effectuer des opérations en lecture seule pour trouver le Clé gérée par AWS DynamoDB dans votre compte.

JSON

```
{
  "Version":"2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
```

```
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

Politique de clé pour une clé gérée par le client

Lorsque vous sélectionnez une clé gérée par le client pour protéger une table DynamoDB, DynamoDB obtient l'autorisation d'utiliser la clé KMS pour le compte du principal qui effectue la sélection. Ce principal, un utilisateur ou un rôle, doit disposer des autorisations requises par DynamoDB sur la clé KMS. Vous pouvez fournir ces autorisations dans une politique de clé, une politique IAM ou un octroi.

DynamoDB requiert au minimum les autorisations suivantes sur une clé gérée par le client :

- kms:Encrypt
- kms:Decrypt
- kms : ReEncrypt * (pour kms: ReEncryptFrom et kms:ReEncryptTo)
- kms : GenerateDataKey * (pour kms : GenerateDataKey et kms : GenerateDataKeyWithoutPlaintext)
- km : DescribeKey
- km : CreateGrant

Par exemple, l'exemple de stratégie de clé suivant fournit uniquement les autorisations requises. La politique a les effets suivants :

- Elle permet à DynamoDB d'utiliser la clé KMS dans les opérations cryptographiques et de créer des octrois, mais seulement lorsque ce service agit pour le compte des principals du compte qui ont l'autorisation d'utiliser DynamoDB. Si les principals spécifiés dans l'instruction de politique n'ont pas l'autorisation d'utiliser DynamoDB, l'appel échoue, même lorsqu'il provient du service DynamoDB.
- La clé de ViaService condition kms : autorise les autorisations uniquement lorsque la demande provient de DynamoDB au nom des principaux acteurs répertoriés dans la déclaration de politique. Ces principals ne peuvent pas appeler ces opérations directement. Notez que la commande valeur `kms:ViaService`, `dynamodb.*.amazonaws.com`, possède un astérisque (*) dans la position Région. DynamoDB doit être autorisé à être indépendant de tout élément Région AWS particulier afin de pouvoir effectuer des appels interrégionaux afin de prendre en charge les tables globales DynamoDB.
- Elle accorde aux administrateurs de clés KMS (utilisateurs qui peuvent endosser le rôle `db-team`) un accès en lecture seule à la clé KMS et l'autorisation de révoquer les octrois, en particulier les octrois requis par DynamoDB pour protéger la table.

Avant d'utiliser un exemple de politique clé, remplacez les exemples de principes par des principes réels provenant de votre.Compte AWS

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
```

```
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

Utilisation d’octrois pour autoriser DynamoDB

En plus des politiques de clé, DynamoDB utilise des octrois pour définir des autorisations sur une clé gérée par un client ou sur la Clé gérée par AWS pour DynamoDB (`aws/dynamodb`). Pour consulter les subventions associées à une clé KMS de votre compte, utilisez l'ListGrantsopération. DynamoDB n’a pas besoin d’octrois, ni d’autorisations supplémentaires, pour utiliser la Clé détenue par AWS et protéger votre table.

DynamoDB utilise les autorisations d’octroi lorsqu’il effectue des tâches de maintenance système en arrière-plan et de protection des données en continu. Il utilise également des octrois pour générer les clés de table.

Chaque octroi est spécifique à une table. Si le compte inclut plusieurs tables chiffrées avec la même clé KMS, il existe un octroi de chaque type pour chaque table. L'autorisation est limitée par le

contexte de chiffrement DynamoDB, qui inclut le nom de la table et l'ID, et elle inclut Compte AWS l'autorisation de retirer la subvention si elle n'est plus nécessaire.

Pour créer les octrois, DynamoDB doit être autorisé à appeler `CreateGrant` au nom de l'utilisateur qui a créé la table chiffrée. En Clés gérées par AWS effet, DynamoDB `kms:CreateGrant` obtient l'autorisation de la politique des clés, qui permet aux utilisateurs du compte d'CreateGrantappeler la clé KMS uniquement lorsque DynamoDB fait la demande au nom d'un utilisateur autorisé.

La politique de clé peut également permettre au compte de révoquer l'octroi sur la clé KMS. Toutefois, si vous révoquez l'octroi sur une table chiffrée active, DynamoDB n'est pas en mesure de protéger ni de maintenir la table.

Client de chiffrement DynamoDB

Un contexte de chiffrement est un ensemble de paires clé-valeur qui contiennent des données non secrètes arbitraires. Lorsque vous incluez un contexte de chiffrement dans une demande de chiffrement de données, lie AWS KMS cryptographiquement le contexte de chiffrement aux données chiffrées. Pour déchiffrer les données, vous devez transmettre le même contexte de chiffrement.

DynamoDB utilise le même contexte de chiffrement dans AWS KMS toutes les opérations cryptographiques. Si vous utilisez une clé gérée par le client ou une Clé gérée par AWS pour protéger votre table DynamoDB, vous pouvez utiliser le contexte de chiffrement pour identifier l'utilisation de la clé KMS dans les enregistrements d'audit et les journaux. Il apparaît également en texte clair dans les journaux, tels que AWS CloudTrailAmazon CloudWatch Logs.

Le contexte de chiffrement peut également être utilisé comme condition d'autorisation dans les politiques et les octrois. DynamoDB utilise le contexte de chiffrement pour limiter les autorisations qui autorisent l'accès à la clé gérée par le client Clé gérée par AWS ou à votre compte et à votre région.

Dans ses demandes adressées àAWS KMS, DynamoDB utilise un contexte de chiffrement avec deux paires clé-valeur.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Table – La première paire clé-valeur identifie la table chiffrée par DynamoDB. La clé est `aws:dynamodb:tableName`. La valeur correspond au nom de la table.

```
"aws:dynamodb:tableName": "<table-name>"
```

Par exemple :

```
"aws:dynamodb:tableName": "Books"
```

- Compte – La deuxième paire clé-valeur identifie le Compte AWS. La clé est `aws:dynamodb:subscriberId`. La valeur correspond à l'ID de compte.

```
"aws:dynamodb:subscriberId": "<account-id>"
```

Par exemple :

```
"aws:dynamodb:subscriberId": "111122223333"
```

Surveillance de l'interaction DynamoDB avec AWS KMS

Si vous utilisez une clé gérée par le client ou une clé Clé gérée par AWSpour protéger vos tables DynamoDB, vous pouvez AWS CloudTrail utiliser les journaux pour suivre les demandes que DynamoDB envoie en votre nom.AWS KMS

Les demandes `GenerateDataKey`, `Decrypt` et `CreateGrant` sont présentées dans cette section. En outre, DynamoDB utilise DescribeKeyune opération pour déterminer si la clé KMS que vous avez sélectionnée existe dans le compte et dans la région. Il utilise également une RetireGrantopération pour supprimer une autorisation lorsque vous supprimez une table.

GenerateDataKey

Lorsque vous activez le chiffrement au repos sur une table, DynamoDB crée une clé de table unique. Il envoie une GenerateDataKeydemande AWS KMS qui spécifie la clé KMS de la table.

L'événement qui enregistre l'opération `GenerateDataKey` est similaire à l'exemple d'événement suivant. L'utilisateur est le compte de service DynamoDB. Les paramètres incluent l'Amazon Resource Name (ARN) de la clé KMS, un spécificateur de clé qui nécessite une clé de 256 bits et le contexte de chiffrement qui identifie la table et le Compte AWS.

```
{
```

```
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```


Decrypt

Lorsque vous accédez à une table DynamoDB chiffrée, DynamoDB a besoin de déchiffrer la clé de table pour pouvoir déchiffrer les clés situées au-dessous dans la hiérarchie. Il déchiffre ensuite les données de la table. Pour déchiffrer la clé de table. DynamoDB envoie une demande AWS KMS de déchiffrement qui indique la clé KMS de la table.

L'événement qui enregistre l'opération `Decrypt` est similaire à l'exemple d'événement suivant. L'utilisateur principal est Compte AWS celui qui accède à la table. Les paramètres incluent la clé de table cryptée (sous forme de blob de texte chiffré) et le contexte de chiffrement qui identifie la table et le.Compte AWSAWS KMS déduit l'ID de la clé KMS à partir du texte chiffré.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
```

```
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CreateGrant

Lorsque vous utilisez une clé gérée par le client ou une Clé gérée par AWS pour protéger votre table DynamoDB, DynamoDB utilise des octrois pour autoriser le service à assurer une protection continue des données, ainsi que des tâches de maintenance et de durabilité. Ces octrois ne sont pas requis sur les Clé détenue par AWS.

Les octrois que DynamoDB crée sont spécifiques à une table. Le principal de la CreateGrantdemande est l'utilisateur qui a créé la table.

L'événement qui enregistre l'opération `CreateGrant` est similaire à l'exemple d'événement suivant. Les paramètres incluent l'Amazon Resource Name (ARN) de la clé KMS de la table, le principal bénéficiaire et le principal de retrait (le service DynamoDB), ainsi que les opérations couvertes par l'octroi. Il inclut également une contrainte qui exige que toutes les opération de chiffrement utilisent le contexte de chiffrement spécifié.

```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
```

```
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
"5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
```

```
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Gestion des tables chiffrées dans DynamoDB

Vous pouvez utiliser le AWS Management Console ou le AWS Command Line Interface(AWS CLI) pour spécifier la clé de chiffrement sur les nouvelles tables et mettre à jour les clés de chiffrement sur les tables existantes dans Amazon DynamoDB.

Rubriques

- Définition de la clé de chiffrement pour une nouvelle table
- Mise à jour d'une clé de chiffrement

Définition de la clé de chiffrement pour une nouvelle table

Suivez ces étapes pour spécifier la clé de chiffrement sur une nouvelle table à l'aide de la console Amazon DynamoDB ou de l'AWS CLI.

Création d'une table chiffrée (console)

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez Créer une table. Comme Nom de la table, entrez **`Music`**. Pour la clé principale, saisissez**`Artist`**, puis, pour la clé de tri, saisissez**`SongTitle`**, ces deux valeurs devant être sous forme de chaîne.

4. Dans Settings (Paramètres), assurez-vous que l'option Customize settings (Personnaliser les paramètres) est sélectionnée.

   > **Note**
   >
   > Si l'option Utiliser les paramètres par défaut est sélectionnée, les tables sont chiffrées Clé détenue par AWS au repos sans frais supplémentaires.

5. Sous Chiffrement au repos, choisissez un type de chiffrement -Clé détenue par AWSClé gérée par AWS, ou une clé gérée par le client.

   - Détenu par Amazon DynamoDB.AWS clé possédée, spécifiquement détenue et gérée par DynamoDB. Aucun frais supplémentaire ne vous est facturé pour l'utilisation de cette clé.
   - AWS clé gérée. alias de clé (`aws/dynamodb`). La clé est enregistrée dans votre compte et est gérée par AWS Key Management Service(AWS KMS).AWS KMS des frais s'appliquent.
   - Stockée dans votre compte, et vous en êtes le propriétaire et le gestionnaire. Clé gérée par le client. La clé est enregistrée dans votre compte et est gérée par AWS Key Management Service(AWS KMS).AWS KMS des frais s'appliquent.

   > **Note**
   >
   > Si vous choisissez de posséder et de gérer votre propre clé, assurez-vous que la politique des clés de KMS est correctement définie. Pour plus d'informations, consultez la politique relative aux clés gérées par le client.

6. Choisissez Create table (Créer une table) pour créer la table chiffrée. Pour confirmer le type de chiffrement, sélectionnez les détails de la table dans l'onglet Overview (Vue d'ensemble) et examinez la section Additional details (Détails supplémentaires).

Création d'une table chiffrée (AWS CLI)

Utilisez le AWS CLI pour créer une table avec la clé par défaut Clé détenue par AWS, la Clé gérée par AWS, ou une clé gérée par le client pour Amazon DynamoDB.

Pour créer une table chiffrée avec la valeur par défaut Clé détenue par AWS

- Créez la table chiffrée `Music` comme suit :

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> Cette table est désormais chiffrée à l'aide de la valeur par défaut Clé détenue par AWS dans le compte de service DynamoDB.

Pour créer une table chiffrée à l'aide de Clé gérée par AWS for DynamoDB

- Créez la table chiffrée `Music` comme suit :

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

Le statut `SSEDescription` de la description de la table est défini sur `ENABLED` et le `SSEType` est `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
```

```
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Pour créer une table chiffrée avec une clé gérée par le client pour DynamoDB

- Créez la table chiffrée `Music` comme suit :

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

  Le statut `SSEDescription` de la description de la table est défini sur `ENABLED` et le `SSEType` est `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Mise à jour d’une clé de chiffrement

Vous pouvez également utiliser la console DynamoDB ou le pour mettre à jour AWS CLI les clés de chiffrement d'une table existante entre une cléClé gérée par AWS, et Clé détenue par AWS une clé gérée par le client à tout moment.

Mise à jour d'une clé de chiffrement (console)

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez la table que vous voulez mettre à jour.
4. Sélectionnez la liste déroulante Actions, puis l'option Update settings (Mettre à jour les paramètres).
5. Accédez à l'onglet Additional settings (Paramètres supplémentaires).
6. Sous Encryption (Chiffrement), choisissez Manage encryption (Gérer le chiffrement).
7. Choisissez un type de chiffrement :

   - Détenue par Amazon DynamoDB. La AWS KMS clé est détenue et gérée par DynamoDB. Aucun frais supplémentaire ne vous est facturé pour l'utilisation de cette clé.
   - AWS clé gérée Alias de clé :`aws/dynamodb`. La clé est enregistrée dans votre compte et est gérée par AWS Key Management Service. (AWS KMS).AWS KMS des frais s'appliquent.
   - Stockée dans votre compte, et vous en êtes le propriétaire et le gestionnaire. La clé est enregistrée dans votre compte et est gérée par AWS Key Management Service. (AWS KMS).AWS KMS des frais s'appliquent.

   > **Note**
   >
   > Si vous choisissez de posséder et de gérer votre propre clé, assurez-vous que la politique des clés de KMS est correctement définie. Pour plus d'informations, consultez Politique de clé pour une clé gérée par le client.

   Choisissez ensuite Enregistrer pour mettre à jour la table chiffrée. Pour vérifier le type de chiffrement, consultez les détails de la table sous l'onglet Présentation.

Mise à jour d'une clé de chiffrement (AWS CLI)

Les exemples suivants illustrent comment mettre à jour une table chiffrée à l'aide de l'AWS CLI.

Pour mettre à jour une table chiffrée avec la valeur par défaut Clé détenue par AWS

- Mettez à jour la table `Music` chiffrée, comme dans l'exemple suivant.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Cette table est désormais chiffrée à l'aide de la valeur par défaut Clé détenue par AWS dans le compte de service DynamoDB.

Pour mettre à jour une table chiffrée avec le Clé gérée par AWS pour DynamoDB

- Mettez à jour la table `Music` chiffrée, comme dans l'exemple suivant.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

Le statut `SSEDescription` de la description de la table est défini sur `ENABLED` et le `SSEType` est `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Pour mettre à jour une table chiffrée avec une clé gérée par le client pour DynamoDB

- Mettez à jour la table `Music` chiffrée, comme dans l'exemple suivant.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

Le statut `SSEDescription` de la description de la table est défini sur `ENABLED` et le `SSEType` est `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Sécurisation des connexions DynamoDB à l'aide de points de terminaison d'un VPC et de politiques IAM

Les connexions sont protégées à la fois entre Amazon DynamoDB et les applications sur site et entre DynamoDB et d'autres ressources au sein de la même région.AWSAWS

Politique requise pour les points de terminaison

Amazon DynamoDB fournit une API DescribeEndpoints qui vous permet d'énumérer les informations relatives aux points de terminaison régionaux. Pour les demandes adressées aux points de terminaison DynamoDB publics, l'API répond quelle que soit la politique IAM DynamoDB configurée, même en cas de refus explicite ou implicite dans la politique de point de terminaison IAM ou d'un VPC. Cela est dû au fait que DynamoDB ignore intentionnellement l'autorisation pour l'API `DescribeEndpoints`.

Pour les demandes provenant d'un point de terminaison d'un VPC, les politiques de point de terminaison IAM et de cloud privé virtuel (VPC) doivent autoriser l'appel d'API `DescribeEndpoints` pour le ou les principaux responsables de la gestion des identités et des accès (IAM) demandeurs à l'aide de l'action IAM `dynamodb:DescribeEndpoints`. Dans le cas contraire, l'accès à l'API `DescribeEndpoints` est refusé.

Voici un exemple de stratégie de point de terminaison.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

Trafic entre les clients de service et sur site et les applications

Vous disposez de deux options de connectivité entre votre réseau privé et AWS :

- Une AWS Site-to-Site VPN connexion. Pour plus d'informations, consultez Présentation de AWS Site-to-Site VPN dans le Guide de l'utilisateur AWS Site-to-Site VPN.
- Une Direct Connect connexion. Pour plus d'informations, consultez Présentation de Direct Connect dans le Guide de l'utilisateur Direct Connect.

L'accès à DynamoDB via le réseau se fait par publication.AWS APIs Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.2. Nous recommandons TLS 1.3. Les clients doivent également prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes. De plus, vous devez signer les demandes à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète, associées à un principal IAM. Vous pouvez également utiliser le service AWS Security Token Service(STS) afin de générer des informations d'identification de sécurité temporaires pour signer les demandes.

Trafic entre des ressources AWS dans la même Région

Un point de terminaison Amazon Virtual Private Cloud (Amazon VPC) pour DynamoDB est une entité logique au sein d'un VPC qui autorise la connectivité uniquement à DynamoDB. Le VPC Amazon achemine les demandes vers DynamoDB et les réponses en retour vers le VPC. Pour plus d'informations, consultez Points de terminaison VPC dans le Guide de l'utilisateur Amazon VPC. Pour des exemples de politiques que vous pouvez utiliser pour contrôler l'accès à partir de points de terminaison d'un VPC, consultez Utilisation de politiques IAM pour contrôler l'accès à DynamoDB.

> **Note**
>
> Les points de terminaison Amazon VPC ne sont pas accessibles via ou.AWS Site-to-Site VPNDirect Connect

Gestion des identités et des accès AWS(IAM) et DynamoDB

Gestion des identités et des accès AWS est un AWS service qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Des administrateurs contrôlent les personnes qui peuvent être authentifiées (connectées) et autorisées (disposant d'autorisations) pour utiliser des ressources Amazon DynamoDB et DynamoDB Accelerator. Vous pouvez utiliser IAM pour gérer les autorisations d'accès et implémenter des politiques de sécurité pour Amazon DynamoDB et DynamoDB Accelerator. IAM est un AWS service que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Gestion des identités et des accès pour Amazon DynamoDB
- Utilisation de conditions de politique IAM pour un contrôle d'accès précis

Gestion des identités et des accès pour Amazon DynamoDB

Gestion des identités et des accès AWS(IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Des administrateurs IAM contrôlent les personnes qui s'authentifient (sont connectées) et sont autorisées (disposent d'autorisations) à utiliser des ressources DynamoDB. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l'accès à l'aide de politiques
- Fonctionnement d'Amazon DynamoDB avec IAM
- Exemples de stratégies basées sur l'identité pour Amazon DynamoDB

- Résolution de problèmes pour l'identité et l'accès Amazon DynamoDB
- Politique IAM pour empêcher l'achat de capacité réservée DynamoDB

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS(IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir Résolution de problèmes pour l'identité et l'accès Amazon DynamoDB)
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir Fonctionnement d'Amazon DynamoDB avec IAM)
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir Exemples de stratégies basées sur l'identité pour Amazon DynamoDB)

Authentification par des identités

L'authentification est la façon dont vous vous connectez àAWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM.Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center(IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS.

Pour l'accès par programmation,AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches

qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder àServices AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède àServices AWS l'aide d'informations d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center.

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder àAWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle d'utilisateur à un rôle IAM (console) ou en appelant une opération d'AWS API AWS CLI ou d'API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès entre comptes, les accès entre services et pour les applications exécutées sur Amazon. EC2 Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l’accès à l’aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant àAWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource.AWSévalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d’informations les documents de politique JSON, consultez Vue d’ensemble des politiques JSON dans le Guide de l’utilisateur IAM.

À l’aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d’aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l’opération.

Politiques basées sur l’identité

Les stratégies basées sur l’identité sont des documents de stratégie d’autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l’identité, consultez Définition d’autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l’utilisateur IAM.

Les politiques basées sur l’identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l’utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l’accès à une ressource spécifique. Vous devez spécifier un principal dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations.
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Fonctionnement d'Amazon DynamoDB avec IAM

Avant d'utiliser IAM pour gérer l'accès à DynamoDB, découvrez les fonctions IAM que vous pouvez utiliser avec DynamoDB.

Fonctionnalité IAM	Support DynamoDB
Politiques basées sur l'identité	Oui
Politiques basées sur les ressources	Oui
Actions de politique	Oui

Fonctionnalité IAM	Support DynamoDB
Ressources de politique	Oui
Clés de condition de politique	Oui
ACLs	Non
ABAC (étiquettes dans les politiques)	Oui
Informations d'identification temporaires	Oui
Autorisations de principal	Oui
Rôles de service	Oui
Rôles liés à un service	Oui

Pour obtenir une vue d'ensemble du fonctionnement de DynamoDB et des AWS autres services avec la plupart des fonctionnalités IAM, AWS consultez la section Services compatibles avec IAM dans le Guide de l'utilisateur IAM.

Politiques basées sur l'identité pour DynamoDB

Prend en charge les politiques basées sur l'identité : oui

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Exemples de stratégies basées sur l'identité pour DynamoDB

Pour voir des exemples de politiques DynamoDB basées sur l'identité, veuillez consulter Exemples de stratégies basées sur l'identité pour Amazon DynamoDB.

Politiques basées sur une ressource dans DynamoDB

Prend en charge les politiques basées sur les ressources : oui

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou.Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Actions de politique pour DynamoDB

Prend en charge les actions de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour afficher la liste des actions DynamoDB, veuillez consulter Actions définies par Amazon DynamoDB dans la Référence de l'autorisation de service.

Les actions de politique dans DynamoDB utilisent le préfixe suivant avant l'action :

```
aws
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Pour voir des exemples de politiques DynamoDB basées sur l'identité, veuillez consulter Exemples de stratégies basées sur l'identité pour Amazon DynamoDB.

Ressources de politique pour DynamoDB

Prend en charge les ressources de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de ressources DynamoDB et ARNs leurs caractéristiques, consultez la section Ressources définies par Amazon DynamoDB dans le Service Authorization Reference. Pour savoir les actions avec lesquelles vous pouvez spécifier l'ARN de chaque ressource, consultez Actions définies par Amazon DynamoDB.

Pour voir des exemples de politiques DynamoDB basées sur l'identité, veuillez consulter Exemples de stratégies basées sur l'identité pour Amazon DynamoDB.

Clés de condition de politique pour DynamoDB

Prend en charge les clés de condition de politique spécifiques au service : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` indique à quel moment les instructions s'exécutent en fonction de critères définis. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM.

Pour afficher la liste des clés de condition DynamoDB, veuillez consulter Clés de condition pour Amazon DynamoDB dans la Référence de l'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez Actions définies par Amazon DynamoDB.

Pour voir des exemples de politiques DynamoDB basées sur l'identité, veuillez consulter Exemples de stratégies basées sur l'identité pour Amazon DynamoDB.

Listes de contrôle d'accès (ACLs) dans DynamoDB

Supports ACLs : Non

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Contrôle d'accès par attributs (ABAC) avec DynamoDB

Prise en charge d'ABAC (balises dans les politiques) : Oui

Le contrôle d'accès par attributs (ABAC) est une stratégie d'autorisation qui définit les autorisations en fonction des attributs appelés balises. Vous pouvez associer des balises aux entités et aux AWS ressources IAM, puis concevoir des politiques ABAC pour autoriser les opérations lorsque la balise du principal correspond à la balise de la ressource.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur ABAC, consultez Définition d'autorisations avec l'autorisation ABAC dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez Utilisation du contrôle d'accès par attributs (ABAC) dans le Guide de l'utilisateur IAM.

Utilisation des informations d'identification temporaires avec DynamoDB

Prend en charge les informations d'identification temporaires : oui

Les informations d'identification temporaires fournissent un accès à court terme aux AWS ressources et sont automatiquement créées lorsque vous utilisez la fédération ou que vous changez de rôle.AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez Informations d'identification de sécurité temporaires dans IAM et Services AWS compatibles avec IAM dans le Guide de l'utilisateur IAM.

Autorisations principales entre services pour DynamoDB

Prend en charge les sessions d'accès direct (FAS) : oui

Les sessions d'accès direct (FAS) utilisent les autorisations du principal appelant et Service AWS, combinées Service AWSà la demande d'envoi de demandes aux services en aval. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez la section Sessions de transmission d'accès.

Fonctions de service pour DynamoDB

Prend en charge les rôles de service : oui

Un rôle de service est un rôle IAM qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

> **Warning**
>
> La modification des autorisations d'une fonction du service peut altérer la fonctionnalité de DynamoDB. Ne modifiez des fonctions du service que quand DynamoDB vous le conseille.

Rôles liés à un service pour DynamoDB

Prend en charge les rôles liés à un service : oui

Un rôle lié à un service est un type de rôle de service lié à un.Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM. Recherchez un service dans le tableau qui inclut un `Yes` dans la colonne Rôle lié à un service. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

Rôles liés au service pris en charge dans DynamoDB

Les rôles liés au service suivants sont pris en charge dans DynamoDB.

- DynamoDB utilise le AWSServiceRoleForDynamoDBReplicationrôle lié à un service pour la réplication des tables globales entre elles.Régions AWS Consultez Sécurité des tables globales DynamoDB pour plus d'informations sur le rôle AWSServiceRoleForDynamoDBReplicationlié à un service.
- DynamoDB Accelerator (DAX) utilise le AWSServiceRoleForrôle DAX lié à un service pour configurer et gérer un cluster DAX. the section called "Utilisation des rôles liés à un service pour DAX"Pour plus d'informations sur le rôle lié au service AWSServiceRoleForDAX, reportez-vous à la section.

Outre ces rôles liés au service DynamoDB, DynamoDB utilise le service Application Auto Scaling pour gérer automatiquement les paramètres de débit sur les tables avec mode de capacité provisionné. Le service Application Auto Scaling utilise le rôle lié au service AWSServiceRoleForApplicationAutoScaling_DynamoDBTablepour gérer les paramètres de débit sur les tables DynamoDB pour lesquelles le dimensionnement automatique est activé. Pour plus d'informations, consultez Rôles liés à un service pour Application Auto Scaling.

Exemples de stratégies basées sur l'identité pour Amazon DynamoDB

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou modifier les ressources DynamoDB. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM.

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, consultez Création de politiques IAM (console) dans le Guide de l'utilisateur IAM.

Pour en savoir plus sur les actions et les types de ressources définis par DynamoDB, y compris le format de ARNs chaque type de ressource, consultez la section Actions, ressources et clés de condition pour Amazon DynamoDB dans la référence d'autorisation de service.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la console DynamoDB
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations
- Utilisation des politiques basées sur une identité avec Amazon DynamoDB

Bonnes pratiques en matière de politiques

Les stratégies basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources DynamoDB dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.
- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder

l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez Validation de politiques avec IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue.Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Sécurisation de l'accès aux API avec MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de la console DynamoDB

Pour accéder à la console Amazon DynamoDB, vous devez disposer d'un ensemble minimum d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher les détails relatifs aux ressources DynamoDB de votre.Compte AWS Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l'AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la console DynamoDB, associez également DynamoDB ou la politique gérée aux `ConsoleAccess` entités `ReadOnly`AWS. Pour plus d'informations, consultez Ajout d'autorisations à un utilisateur dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les

autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Utilisation des politiques basées sur une identité avec Amazon DynamoDB

Cette rubrique traite de l'utilisation des politiques basées sur l'identitéGestion des identités et des accès AWS(IAM) avec Amazon DynamoDB et fournit des exemples. Les exemples montrent comment un administrateur de comptes peut attacher des politiques d’autorisations à des identités

IAM (utilisateurs, groupes et rôles), et ainsi accorder des autorisations d'effectuer des opérations sur des ressources Amazon DynamoDB.

Les sections de cette rubrique couvrent les sujets suivants :

- Autorisations IAM requises pour utiliser la console Amazon DynamoDB
- AWS politiques IAM gérées (prédéfinies) pour Amazon DynamoDB
- Exemples de politiques gérées par le client

Un exemple de politique d'autorisation est exposé ci-dessous.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
    ]
}
```

La politique précédente comporte une déclaration qui accorde des autorisations pour trois actions DynamoDB `dynamodb:DescribeTable` (`dynamodb:Query`,, `dynamodb:Scan` et) sur une table de `us-west-2`AWS la région, qui appartient au compte spécifié par AWS. *`account-id`* L'Amazon Resource Name (ARN) dans la valeur `Resource` spécifie la table à laquelle les autorisations s'appliquent.

Autorisations IAM requises pour utiliser la console Amazon DynamoDB

Pour utiliser la console DynamoDB, un utilisateur doit disposer d'un ensemble minimal d'autorisations lui permettant d'utiliser les ressources DynamoDB de son AWS compte. Outre ces autorisations DynamoDB, la console nécessite d'autres autorisations :

- CloudWatch Autorisations Amazon pour afficher les statistiques et les graphiques.
- AWS Data Pipeline autorisations d'exportation et d'importation de données DynamoDB.
- Gestion des identités et des accès AWS autorisations d'accès aux rôles nécessaires pour les exportations et les importations.
- Amazon Simple Notification Service est autorisé à vous avertir chaque fois qu'une CloudWatch alarme est déclenchée.
- AWS Lambda autorisations pour traiter les enregistrements DynamoDB Streams.

Si vous créez une politique IAM plus restrictive que les autorisations minimales requises, la console ne fonctionnera pas comme prévu pour les utilisateurs dotés de cette politique IAM. Pour garantir que ces utilisateurs peuvent toujours utiliser la console DynamoDB, associez également la politique gérée à `AmazonDynamoDBReadOnlyAccess`AWS l'utilisateur, comme décrit dans. AWS politiques IAM gérées (prédéfinies) pour Amazon DynamoDB

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l'API Amazon DynamoDB.

> **Note**
>
> Si vous faites référence à un point de terminaison VPC, vous devez également autoriser l'appel d' DescribeEndpoints API pour le ou les principaux IAM demandeurs avec l'action IAM (dynamodb :). DescribeEndpoints Pour de plus amples informations, veuillez consulter Politique requise pour les points de terminaison.

AWS politiques IAM gérées (prédéfinies) pour Amazon DynamoDB

AWS répond à certains cas d'utilisation courants en fournissant des politiques IAM autonomes créées et administrées par.AWS Ces politiques AWS gérées accordent les autorisations nécessaires pour les cas d'utilisation courants afin que vous puissiez éviter d'avoir à rechercher les autorisations nécessaires. Pour plus d'informations, consultez Politiques gérées par AWS dans le Guide de l'utilisateur IAM.

Les politiques AWS gérées suivantes, que vous pouvez associer aux utilisateurs de votre compte, sont spécifiques à DynamoDB et sont regroupées par scénario d'utilisation :

- AmazonDynamoDBReadOnlyAccess— Accorde un accès en lecture seule aux ressources DynamoDB via le.AWS Management Console
- AmazonDynamoDBFullAccès : accorde un accès complet aux ressources DynamoDB via le.AWS Management Console

Vous pouvez consulter ces politiques d'autorisations AWS gérées en vous connectant à la console IAM et en y recherchant des politiques spécifiques.

> **Important**
>
> La bonne pratique consiste à créer des politiques IAM personnalisées qui accordent le moindre privilège aux utilisateurs, rôles ou groupes IAM qui en ont besoin.

Exemples de politiques gérées par le client

Cette section contient des exemples de politiques qui accordent des autorisations pour diverses actions DynamoDB. Ces politiques fonctionnent lorsque vous utilisez AWS SDKs ou le AWS CLI. Lorsque vous utilisez la console, vous devez accorder des autorisations supplémentaires spécifiques de la console. Pour de plus amples informations, veuillez consulter Autorisations IAM requises pour utiliser la console Amazon DynamoDB.

> **Note**
>
> Tous les exemples de politique suivants utilisent l'une des AWS régions et contiennent des noms de comptes IDs et de tables fictifs.

Exemples :

- Politique IAM pour accorder des autorisations à toutes les actions DynamoDB sur une table
- Politique IAM pour accorder des autorisations en lecture seule sur des éléments dans une table DynamoDB
- Politique IAM pour accorder l’accès à une table DynamoDB spécifique et à ses index

- Politique IAM pour lire, écrire, mettre à jour et supprimer l'accès sur une table DynamoDB
- Politique IAM pour séparer les environnements DynamoDB dans le même compte AWS
- Politique IAM pour empêcher l'achat de capacité réservée DynamoDB
- Politique IAM pour accorder un accès en lecture pour un flux DynamoDB uniquement (pas pour la table)
- Politique IAM permettant à une AWS Lambda fonction d'accéder aux enregistrements de flux DynamoDB
- Politique IAM pour l'accès en lecture et écriture à un cluster DynamoDB Accelerator (DAX)

Le Guide de l'utilisateur IAM inclut trois exemples DynamoDB supplémentaires :

- Amazon DynamoDB : autorise l'accès à une table spécifique
- Amazon DynamoDB : autorise l'accès à des colonnes spécifiques
- Amazon DynamoDB : autorise l'accès de niveau ligne à DynamoDB en fonction de l'ID Amazon Cognito

Politique IAM pour accorder des autorisations à toutes les actions DynamoDB sur une table

La politique suivante accorde des autorisations pour toutes les actions DynamoDB sur une table nommée `Books`. L'ARN de ressource spécifié dans le `Resource` identifie une table dans une AWS région spécifique. Si vous remplacez le nom de table `Books` dans l'ARN `Resource` par un caractère générique (*), toutes les actions DynamoDB sont autorisées sur toutes les tables du compte. Considérez attentivement les implications possibles en matière de sécurité avant d'utiliser un caractère générique sur cette politique ou toute politique IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
```

```
    ]
}
```

> **Note**
>
> Voici un exemple d'utilisation d'un caractère générique (*) pour autoriser toutes les actions, y compris l'administration, les opérations de données, la surveillance et l'achat de capacité réservée DynamoDB. Au lieu de cela, une bonne pratique consiste à spécifier explicitement chaque action à autoriser, et uniquement ce dont cet utilisateur, ce rôle ou ce groupe ont besoin.

Politique IAM pour accorder des autorisations en lecture seule sur des éléments dans une table DynamoDB

La politique d'autorisations suivante accorde des autorisations pour les actions DynamoDB `GetItem`, `BatchGetItem`, `Scan`, `Query` et `ConditionCheckItem` uniquement, et définit ainsi un accès en lecture seule sur la table `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Politique IAM pour accorder l'accès à une table DynamoDB spécifique et à ses index

La politique suivante accorde des autorisations pour effectuer des actions de modification de données sur une table DynamoDB appelée `Books` et tous ses index. Pour plus d'informations sur le fonctionnement des index, consultez Amélioration de l'accès aux données avec les index secondaires dans DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Politique IAM pour lire, écrire, mettre à jour et supprimer l'accès sur une table DynamoDB

Utilisez cette politique si vous devez autoriser votre application à créer, lire, mettre à jour et supprimer des données dans des tables, index et flux Amazon DynamoDB. Remplacez le nom de la AWS région, votre identifiant de compte et le nom de la table ou le caractère générique (*) le cas échéant.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
```

```
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Pour étendre cette politique à toutes les tables DynamoDB de AWS toutes les régions pour ce compte, utilisez un caractère générique (*) pour la région et le nom de la table. Par exemple :

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

Politique IAM pour séparer les environnements DynamoDB dans le même compte AWS

Supposons que vous ayez des environnements séparés gérant chacun sa propre version d'une table nommée `ProductCatalog`. Si vous créez deux `ProductCatalog` tables dans le même AWS compte, le travail dans un environnement peut affecter l'autre environnement en raison de la manière dont les autorisations sont configurées. Par exemple, les quotas relatifs au nombre d'opérations simultanées sur le plan de contrôle (telles que`CreateTable`) sont définis au niveau du AWS compte.

Par conséquent, chaque action effectuée dans un environnement réduit le nombre d'opérations disponibles dans l'autre. Il existe également un risque que le code d'un environnement puisse accéder accidentellement aux tables de l'autre.

> **Note**
>
> Si vous souhaitez séparer les charges de travail de test et de production afin de contrôler les répercussions potentielles d'un incident, une bonne pratique consiste à créer des comptes AWS distincts pour les charges de travail de test et de production. Pour plus d'informations, consultez Gestion et séparation de comptes AWS.

Supposons également que vous ayez deux développeurs, Amit et Alice, qui testent la table `ProductCatalog`. Au lieu que chaque développeur ait besoin d'un AWS compte distinct, vos développeurs peuvent partager le même AWS compte de test. Dans ce compte de test, vous

pouvez créer une copie de la même table pour que chaque développeur puisse y travailler, comme `Alice_ProductCatalog` et `Amit_ProductCatalog`. Dans ce cas, vous pouvez créer les utilisateurs Alice et Amit dans le AWS compte que vous avez créé pour l'environnement de test. Vous pouvez ensuite accorder à ces utilisateurs l'autorisation d'effectuer des actions DynamoDB sur les tables qu'ils possèdent.

Pour accorder ces autorisations utilisateur IAM, vous pouvez procéder de l'une des façons suivantes :

- Créez une politique distincte pour chaque utilisateur, puis attachez chaque politique à son utilisateur séparément. Par exemple, vous pouvez attacher la politique suivante à l'utilisateur Alice pour l'autoriser à accéder à toutes les actions DynamoDB sur la table `Alice_ProductCatalog` :

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
              "dynamodb:DescribeContinuousBackups",
              "dynamodb:BatchGetItem",
              "dynamodb:UpdateTimeToLive",
              "dynamodb:BatchWriteItem",
              "dynamodb:ConditionCheckItem",
              "dynamodb:UntagResource",
              "dynamodb:PutItem",
              "dynamodb:Scan",
              "dynamodb:Query",
```

```
                "dynamodb:UpdateItem",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:UpdateTable",
                "dynamodb:DescribeTableReplicaAutoScaling",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:DescribeLimits",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

Ensuite, vous pouvez créer une politique similaire avec une autre ressource (table `Amit_ProductCatalog`) pour l'utilisateur Amit.

- Au lieu d'attacher les politiques à des utilisateurs individuels, vous pouvez utiliser les variables des politiques IAM pour écrire une seule politique et l'attacher à un groupe. Vous devez créer un groupe et, pour cet exemple, y ajouter les utilisateurs Alice et Amit. L'exemple suivant accorde des autorisations pour exécuter toutes les actions DynamoDB sur la table `${aws:username}_ProductCatalog`. La variable de politique `${aws:username}` est remplacée par le nom utilisateur du demandeur lors de l'évaluation de la politique. Par exemple, si Alice envoie une demande pour ajouter un élément, l'action est autorisée uniquement si Alice ajoute les éléments à la table `Alice_ProductCatalog`.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
```

```
                "dynamodb:BatchWriteItem",
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> Lorsque vous utilisez des variables de politique IAM, vous devez spécifier explicitement la version `2012-10-17` du langage de politique IAM dans la politique. La version par défaut du langage de politique IAM (`2008-10-17`) ne prend pas en charge les variables de politique.

Au lieu d'identifier une table spécifique en tant que ressource comme vous le feriez normalement, vous pouvez utiliser un caractère générique (*) pour accorder des autorisations sur toutes les tables dont le nom est préfixé avec le nom de l'utilisateur qui effectue la demande, comme illustré dans l'exemple suivant.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Politique IAM pour empêcher l'achat de capacité réservée DynamoDB

Avec une capacité réservée Amazon DynamoDB, vous payez un droit initial unique, et vous engagez à payer pour un niveau d'utilisation minimal, avec à la clé la réalisation d'économies considérables au fil du temps. Vous pouvez utiliser le AWS Management Console pour consulter et acheter des capacités réservées. Cependant, il se peut que vous ne souhaitiez pas que tous les utilisateurs au sein de votre organisation puissent acheter de la capacité réservée. Pour plus d'informations sur la capacité réservée, consultez Tarification Amazon DynamoDB.

DynamoDB fournit les opérations d'API suivantes pour contrôler l'accès à la gestion de la capacité réservée :

- `dynamodb:DescribeReservedCapacity` – Renvoie les achats de capacité réservée qui sont actuellement en vigueur.
- `dynamodb:DescribeReservedCapacityOfferings` – Renvoie des détails sur les plans de capacité réservée actuellement proposés par AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Effectue un achat réel de capacité réservée.

Il AWS Management Console utilise ces actions d'API pour afficher les informations sur les capacités réservées et effectuer des achats. Vous ne pouvez pas appeler ces opérations à partir d'un programme d'application, car elles ne sont accessibles qu'à partir de la console. Cependant, vous pouvez autoriser ou rejeter l'accès à ces opérations dans une politique d'autorisations IAM.

La politique suivante permet aux utilisateurs de consulter les achats et les offres de capacité réservée en utilisant le AWS Management Console, mais les nouveaux achats sont refusés.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
```

```
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Notez que cette politique utilise le caractère générique (*) pour décrire les autorisations pour tous, et pour rejeter l'achat de capacité réservée DynamoDB pour tous.

Politique IAM pour accorder un accès en lecture pour un flux DynamoDB uniquement (pas pour la table)

Lorsque vous activez DynamoDB Streams sur une table, des informations sont capturées sur chaque modification apportée à des éléments dans celle-ci. Pour de plus amples informations, veuillez consulter Modifier la récupération de données pour DynamoDB Streams.

Dans certains cas, vous pouvez être amené à empêcher une application de lire les données d'une table DynamoDB, tout en autorisant l'accès aux flux de celle-ci. Par exemple, vous pouvez configurer AWS Lambda pour interroger un flux et appeler une fonction Lambda lorsque des mises à jour d'éléments sont détectées, puis effectuer un traitement supplémentaire.

Les actions suivantes sont disponibles pour contrôler l'accès à DynamoDB Streams :

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

L'exemple de politique suivant accorde aux utilisateurs des autorisations d'accès aux flux d'une table nommée `GameScores`. Le caractère générique final (*) dans l'ARN correspond à tout flux associé à la table.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Notez que cette politique donne accès aux flux de la table `GameScores`, mais pas à la table proprement dite.

Politique IAM permettant à une AWS Lambda fonction d'accéder aux enregistrements de flux DynamoDB

Si vous souhaitez que certaines actions soient effectuées en fonction des événements d'un flux DynamoDB, vous pouvez écrire AWS Lambda une fonction déclenchée par ces événements. Une fonction Lambda telle que celle-ci a besoin d'autorisations pour lire les données d'un flux DynamoDB. Pour plus d'informations sur l'utilisation de Lambda avec DynamoDB Streams, consultez Streams et déclencheurs DynamoDB AWS Lambda.

Pour accorder des autorisations à Lambda, utilisez la politique d'autorisations associée au rôle IAM de la fonction Lambda (également appelée rôle d'exécution). Spécifiez cette politique lorsque vous créez la fonction Lambda.

Par exemple, vous pouvez associer la politique d'autorisations et le rôle d'exécution suivants pour accorder les autorisations Lambda nécessaires pour exécuter les actions DynamoDB Streams répertoriées.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Pour plus d'informations, consultez Autorisations AWS Lambda dans le Manuel du développeur AWS Lambda.

Politique IAM pour l'accès en lecture et écriture à un cluster DynamoDB Accelerator (DAX)

La politique suivante autorise l'accès en lecture, en écriture, en mise à jour et en suppression à un cluster DynamoDB Accelerator (DAX), mais pas à la table DynamoDB associée. Pour utiliser cette politique, remplacez le nom de la AWS région, votre identifiant de compte et le nom de votre cluster DAX.

> **Note**
>
> Cette politique permet d'accéder au cluster DAX, mais pas à la table DynamoDB associée. Assurez-vous que votre cluster DAX dispose de la politique appropriée pour effectuer ces mêmes opérations sur la table DynamoDB en votre nom.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

Pour étendre cette politique afin de couvrir l'accès au DAX pour toutes les AWS régions pour un compte, utilisez un caractère générique (*) pour le nom de la région.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Résolution de problèmes pour l’identité et l’accès Amazon DynamoDB

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec DynamoDB et IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans DynamoDB
- Je ne suis pas autorisé à effectuer iam : PassRole

- Je souhaite autoriser des personnes extérieures à moi Compte AWSà accéder à mes ressources DynamoDB

Je ne suis pas autorisé à effectuer une action dans DynamoDB

S'il vous AWS Management Console indique que vous n'êtes pas autorisé à effectuer une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe.

L'exemple d'erreur suivant se produit quand l'utilisateur `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `aws:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses politiques pour lui permettre d'accéder à la ressource *`my-example-widget`* à l'aide de l'action `aws:`*`GetWidget`*.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à DynamoDB.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans DynamoDB. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite autoriser des personnes extérieures à moi Compte AWSà accéder à mes ressources DynamoDB

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si DynamoDB prend en charge ces fonctionnalités, consultez Fonctionnement d'Amazon DynamoDB avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Politique IAM pour empêcher l'achat de capacité réservée DynamoDB

Avec une capacité réservée Amazon DynamoDB, vous payez un droit initial unique, et vous engagez à payer pour un niveau d'utilisation minimal, avec à la clé la réalisation d'économies considérables au fil du temps. Vous pouvez utiliser le AWS Management Console pour consulter et acheter des capacités réservées. Cependant, il se peut que vous ne souhaitiez pas que tous les utilisateurs au sein de votre organisation puissent acheter de la capacité réservée. Pour plus d'informations sur la capacité réservée, consultez Tarification Amazon DynamoDB.

DynamoDB fournit les opérations d'API suivantes pour contrôler l'accès à la gestion de la capacité réservée :

- `dynamodb:DescribeReservedCapacity` – Renvoie les achats de capacité réservée qui sont actuellement en vigueur.
- `dynamodb:DescribeReservedCapacityOfferings` – Renvoie des détails sur les plans de capacité réservée actuellement proposés par AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Effectue un achat réel de capacité réservée.

Il AWS Management Console utilise ces actions d'API pour afficher les informations sur les capacités réservées et effectuer des achats. Vous ne pouvez pas appeler ces opérations à partir d'un programme d'application, car elles ne sont accessibles qu'à partir de la console. Cependant, vous pouvez autoriser ou rejeter l'accès à ces opérations dans une politique d'autorisations IAM.

La politique suivante permet aux utilisateurs de consulter les achats et les offres de capacité réservée en utilisant le AWS Management Console, mais les nouveaux achats sont refusés.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
```

```
}
```

Notez que cette politique utilise le caractère générique (*) pour décrire les autorisations pour tous, et pour rejeter l'achat de capacité réservée DynamoDB pour tous.

Utilisation de conditions de politique IAM pour un contrôle d'accès précis

Lorsque vous accordez des autorisations dans DynamoDB, vous pouvez spécifier des conditions pour déterminer comment une politique d'autorisation doit prendre effet.

Présentation de

Dans DynamoDB, vous avez la possibilité de spécifier des conditions lorsque vous accordez des autorisations à l'aide d'une politique IAM (voir Gestion des identités et des accès pour Amazon DynamoDB). Par exemple, vous pouvez effectuer les actions suivantes :

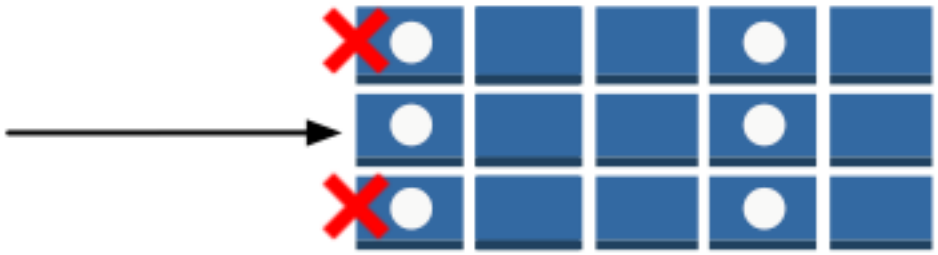
- Accordez des autorisations pour permettre aux utilisateurs d'accéder en lecture seule à certains éléments et attributs dans une table ou un index secondaire.
- Accordez des autorisations pour permettre aux utilisateurs d'accéder en écriture seule à certains attributs d'une table, en fonction de l'identité de cet utilisateur.

Dans DynamoDB, vous pouvez spécifier des conditions dans une politique IAM à l'aide de clés de condition, comme illustré dans le cas d'utilisation de la section suivante.

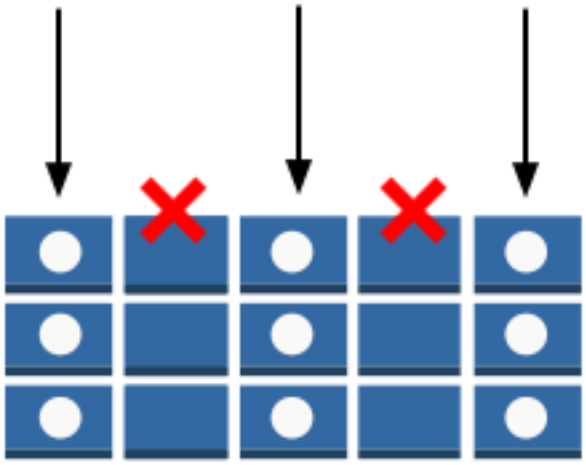
Cas d'utilisation des autorisations

Outre le contrôle de l'accès aux actions d'API DynamoDB, vous pouvez contrôler l'accès à des éléments et attributs de données individuels. Par exemple, vous pouvez effectuer les opérations suivantes :

- Accorder des autorisations sur une table, mais restreindre l'accès à des éléments spécifiques de cette table en fonction de certaines valeurs de clé primaire. A titre d'exemple, une application de réseau social pour les jeux, où les données de jeu enregistrées de tous les utilisateurs sont stockées dans une seule table, mais où aucun utilisateur ne peut accéder aux éléments de données qu'il ne possède pas, comme illustré ci-après :



- Masquer les informations afin que seul un sous-ensemble d'attributs soit visible de l'utilisateur. Un exemple peut être une application qui affiche les données de vol des aéroports proches, en fonction de l'emplacement de l'utilisateur. Les noms des compagnies aériennes, les heures d'arrivée et de départ, et les numéros de vol sont tous affichés. Cependant, les attributs tels que les noms des pilotes ou le nombre de passagers sont masqués, comme illustré ci-après :



Pour mettre en place ce genre de contrôle précis des accès, vous écrivez une politique d'autorisations IAM qui spécifie les conditions d'accès aux informations d'identification de sécurité et aux autorisations associées. Vous appliquez la stratégie aux utilisateurs, groupes ou rôles que vous créez à l'aide de la console IAM. Votre politique IAM peut restreindre l'accès à des éléments individuels d'une table, aux attributs dans ces éléments, ou aux deux à la fois.

Le cas échéant, vous pouvez utiliser la fédération d'identité web pour contrôler l'accès par les utilisateurs authentifiés par Login with Amazon, Facebook ou Google. Pour de plus amples informations, veuillez consulter Utilisation de la fédération d'identité web.

Vous utilisez l'élément IAM `Condition` pour mettre en place une politique de contrôle d'accès détaillée. En ajoutant un élément `Condition` à une politique d'autorisations, vous pouvez autoriser ou rejeter l'accès à des éléments et attributs dans des index et des tables DynamoDB, en fonction de vos besoins particuliers.

Par exemple, imaginons une application de jeux pour appareils mobiles à partir de laquelle les utilisateurs peuvent sélectionner une grande diversité de jeux. L'application utilise une table nommée DynamoDB nommée `GameScores` pour suivre les scores élevés et d'autres données utilisateur. Chaque élément de la table est identifié de façon unique par un ID d'utilisateur et le nom du jeu auquel l'utilisateur a joué. La table `GameScores` possède une clé primaire composée d'une clé de partition (`UserId`) et d'une clé de tri (`GameTitle`). Les utilisateurs ne peuvent accéder qu'aux données de jeu associées à leur ID utilisateur. Un utilisateur qui souhaite jouer une partie doit appartenir à un rôle IAM nommé `GameRole` auquel une politique de sécurité est attachée.

Pour gérer les autorisations utilisateur de cette application, vous pouvez écrire une politique d'autorisations telle que la suivante :

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
```

```
                ],
                "dynamodb:Attributes":[
                   "UserId",
                   "GameTitle",
                   "Wins",
                   "Losses",
                   "TopScore",
                   "TopScoreDateTime"
                ]
             },
             "StringEqualsIfExists":{
                "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
             }
          }
       }
    ]
}
```

En plus de l'octroi d'autorisations pour des actions DynamoDB spécifiques (élément `Action`) sur la table `GameScores` (élément `Resource`), l'élément `Condition` utilise les clés de condition suivantes spécifiques de DynamoDB qui limitent les autorisations comme suit :

- `dynamodb:LeadingKeys` – Cette clé de condition permet aux utilisateurs d'accéder uniquement aux éléments dans lesquels la valeur de clé de partition correspond à leur ID utilisateur. Cet ID, `${www.amazon.com:user_id}`, est une variable de substitution. Pour plus d'informations sur les variables de substitution, consultez Utilisation de la fédération d'identité web.
- `dynamodb:Attributes` – Cette clé de condition limite l'accès aux attributs spécifiés afin que seules les actions figurant dans la politique d'autorisations puissent renvoyer des valeurs pour ces attributs. En outre, la clause `StringEqualsIfExists` garantit que l'application fournisse toujours une liste d'attributs spécifiques sur lesquels agir et que l'application ne peut pas demander tous les attributs.

Quand une politique IAM est évaluée, le résultat est toujours true (accès autorisé) ou false (accès refusé). Si une partie d'un élément `Condition` a la valeur false, l'ensemble de la politique a la valeur false et l'accès est refusé.

> **Important**
>
> Si vous utilisez `dynamodb:Attributes`, vous devez spécifier les noms de tous les attributs de clé primaire et de clé d'index de la table et tous les index secondaires répertoriés dans la politique. Sinon, DynamoDB ne peut pas utiliser ces attributs de clé pour exécuter l'action demandée.

Les documents de la politique IAM peuvent contenir uniquement les caractères Unicode suivants : tabulation horizontale (U+0009), saut de ligne (U+000A), retour chariot (U+000D) et caractères de la plage U+0020 à U+00FF.

Spécification de conditions : utilisation de clés de condition

AWS fournit un ensemble de clés de condition prédéfinies (clés AWS de condition étendues) pour tous les AWS services qui prennent en charge le contrôle d'accès IAM. Par exemple, vous pouvez utiliser la clé de condition `aws:SourceIp` pour vérifier l'adresse IP du demandeur avant de permettre l'exécution d'une action. Pour plus d'informations et une liste des touches « wide », consultez la AWS section Clés disponibles pour les conditions dans le guide de l'utilisateur IAM.

Le tableau suivant illustre les clés de condition spécifiques du service DynamoDB qui s'appliquent à DynamoDB

Clé de condition DynamoDB	Description
`dynamodb:LeadingKeys`	Représente le premier attribut de clé d'une table, c'est-à-dire la clé de partition. Le nom de la clé `LeadingKeys` est pluriel, même si la clé est utilisée avec des actions à élément unique. En outre, vous devez utiliser le modificateur `ForAllValues` lorsque vous utilisez `LeadingKeys` dans une condition.
`dynamodb:FirstPartitionKeyValues`	Représente le premier attribut clé d'une table, en d'autres termes, la première clé de partition. Le nom de la clé `FirstPartitionKeyValues` est pluriel, même si la clé est utilisée avec des actions à élément unique. En outre, vous devez utiliser le `ForAllValues` modificateur lorsque vous l'utilisez `FirstPartitionKeyValues`

Clé de condition DynamoDB	Description
	dans une condition. `FirstPartitionKeyValues` et `LeadingKeys` peut être utilisé comme échangeable.
`dynamodb:SecondPartitionKeyValues`	Similaire à `dynamodb:FirstPartitionKeyValues`. Représente la deuxième clé de partition des ressources. Le nom de la clé `SecondPartitionKeyValues` est pluriel, même si la clé est utilisée avec des actions à élément unique.
`dynamodb:ThirdPartitionKeyValues`	Similaire à `dynamodb:FirstPartitionKeyValues`. Représente la troisième clé de partition des ressources. Le nom de la clé `ThirdPartitionKeyValues` est pluriel, même si la clé est utilisée avec des actions à élément unique.
`dynamodb:FourthPartitionKeyValues`	Similaire à `dynamodb:FirstPartitionKeyValues`. Représente la quatrième clé de partition des ressources. Le nom de la clé `FourthPartitionKeyValues` est pluriel, même si la clé est utilisée avec des actions à élément unique.
`dynamodb:Select`	Représente le paramètre `Select` d'une demande. `Select` peut avoir l'une des valeurs suivantes : • `ALL_ATTRIBUTES` • `ALL_PROJECTED_ATTRIBUTES` • `SPECIFIC_ATTRIBUTES` • `COUNT`

Clé de condition DynamoDB	Description
`dynamodb:Attributes`	Représente une liste des attributs de niveau supérieur auxquels une demande accède. Une demande accède à un attribut de niveau supérieur si celui-ci, ou tout attribut imbriqué qu'il contient, est spécifié dans les paramètres de la demande. Par exemple, dans le `GetItem` cas d'une demande spécifiant un `ProjectionExpression` de`"Name, Address.City"` , la `dynamodb:Attributes` liste inclurait « Nom » et « Adresse ». Si le `Attributes` paramètre est énuméré dans une politique de contrôle d'accès précise, envisagez également de le restreindre `ReturnValues` et de définir des `Select` paramètres afin de garantir un accès restreint aux attributs spécifiés dans le cadre de plusieurs actions d'API telles que `GetItem`, `Query`, et. `Scan`
`dynamodb:ReturnValues`	Représente le `ReturnValues` paramètre d'une demande. Selon l'action de l'API, il `ReturnValues` peut s'agir de l'une des valeurs suivantes : • `ALL_OLD` • `UPDATED_OLD` • `ALL_NEW` • `UPDATED_NEW` • `NONE`
`dynamodb:ReturnConsumedCapacity`	Représente le paramètre `ReturnConsumedCapacity` d'une demande. `ReturnConsumedCapacity` peut avoir l'une des valeurs suivantes : • `TOTAL` • `NONE`

Limitation de l'accès utilisateur

La plupart des politiques d'autorisation IAM permettent aux utilisateurs d'accéder uniquement aux éléments d'une table où la valeur de la clé de partition correspond à l'identifiant de l'utilisateur.

Par exemple, l'application de jeu précédente limite l'accès de cette façon afin que les utilisateurs ne puissent accéder qu'aux données de jeu associées à leur ID utilisateur. Les variables de substitution IAM `${www.amazon.com:user_id}`, `${graph.facebook.com:id}` et `${accounts.google.com:sub}` contiennent des identifiants utilisateur pour Login with Amazon, Facebook et Google. Pour savoir comment une application se connecte à l'un de ces fournisseurs d'identité et obtient ces identificateurs, consultez Utilisation de la fédération d'identité web.

Important

Le contrôle d'accès précis n'est pas pris en charge pour restreindre la réplication des tables globales. L'application de conditions de politique pour un contrôle d'accès précis aux principaux de service DynamoDB ou aux rôles liés à un service utilisés pour la réplication de tables globales peut interrompre la réplication au sein d'une table globale.

Note

Chacun des exemples de la section suivante définit la clause `Effect` sur `Allow` et spécifie uniquement les actions, ressources et paramètres autorisés. L'accès est autorisé uniquement à ce qui est explicitement indiqué dans la politique IAM.
Dans certains cas, il est possible de réécrire ces politiques afin qu'elles soient basées sur le refus (autrement dit, définition de la clause `Effect` sur `Deny` et inversion de toute la logique de la politique). Cependant, nous vous recommandons d'éviter d'utiliser des politiques basées sur un rejet avec DynamoDB, car elles sont difficiles à écrire correctement en comparaison des politiques basées sur une autorisation. En outre, de futures modifications de l'API DynamoDB (ou des modifications apportées à des entrées d'API existantes) peuvent rendre une politique basée sur un rejet inefficace.

Exemples de politique : utilisation de conditions pour un contrôle d'accès détaillé

Cette section présente plusieurs politiques d'implémentation d'un contrôle précis des accès aux tables et index DynamoDB.

Note

Tous les exemples utilisent la région us-west-2 et contiennent un compte fictif. IDs

La vidéo ci-dessous explique le contrôle des accès précis dans DynamoDB à l'aide des conditions de politique IAM.

1 : accorder des autorisations qui limitent l'accès aux éléments avec une valeur de clé de partition spécifique

La politique d'autorisation suivante accorde les autorisations qui permettent un ensemble d'actions DynamoDB sur la table `GamesScore`. Elle utilise la clé de condition `dynamodb:LeadingKeys` pour limiter les actions utilisateur uniquement sur les éléments dont la valeur de la clé de partition `UserID` correspond à la connexion avec l'ID utilisateur Login with Amazon unique pour cette application.

> **Important**
>
> La liste des actions n'inclut pas les autorisations pour `Scan`, car `Scan` retourne tous les éléments, quelles que soient les clés principales.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
```

```
            ]
         }
      }
   }
}
```

> **Note**
>
> Lorsque vous utilisez des variables de politique, vous devez spécifier explicitement la version `2012-10-17` de la politique. La version par défaut du langage de la politique d'accès, `2008-10-17`, ne prend pas en charge les variables de politique.

Pour implémenter l'accès en lecture seule, vous pouvez supprimer les actions susceptibles de modifier les données. Dans la politique suivante, seules les actions qui fournissent l'accès en lecture seule sont incluses dans la condition.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
```

```
         }
      }
   ]
}
```

> **Important**
>
> Si vous utilisez `dynamodb:Attributes`, vous devez spécifier les noms de tous les attributs de clé primaire et de clé d'index pour la table et tous les index secondaires répertoriés dans la politique. Sinon, DynamoDB ne peut pas utiliser ces attributs de clé pour exécuter l'action demandée.

2 : accorder les autorisations qui limitent l'accès aux attributs spécifiques d'une table

La politique d'autorisation suivante n'autorise l'accès qu'à deux attributs spécifiques d'une table en ajoutant la clé de condition `dynamodb:Attributes`. Ces attributs peuvent être lus, écrits ou évalués dans une écriture conditionnelle ou un filtre d'analyse.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
```

```
                "UserId",
                "TopScore"
            ]
         },
         "StringEqualsIfExists":{
            "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
            "dynamodb:ReturnValues":[
                "NONE",
                "UPDATED_OLD",
                "UPDATED_NEW"
            ]
         }
      }
   ]
}
```

> **Note**
>
> La politique adopte une approche de liste d'autorisations (ou liste verte), qui autorise l'accès à un ensemble nommé d'attributs. Vous pouvez écrire une politique équivalente qui refuse à la place l'accès aux autres attributs. Nous ne recommandons pas cette approche de liste de rejet (ou liste rouge). Les utilisateurs peuvent déterminer le nom de ces attributs refusés en suivant le principe du moindre privilège, comme expliqué dans Wikipedia à l'adresse http://en.wikipedia. org/wiki/Principle_of_least_privilege, et utilisez une approche de liste d'autorisation pour énumérer toutes les valeurs autorisées, plutôt que de spécifier les attributs refusés.

Cette politique n'autorise pas `PutItem`, `DeleteItem` ou `BatchWriteItem`. Ces actions remplacent toujours la totalité de l'élément précédent, ce qui permet aux utilisateurs de supprimer les valeurs précédentes des attributs auxquels ils ne sont pas autorisés à accéder.

La clause `StringEqualsIfExists` de la politique d'autorisation garantit ce qui suit :

- Si l'utilisateur spécifie le paramètre `Select`, sa valeur doit être `SPECIFIC_ATTRIBUTES`. Cette exigence empêche l'action d'API de retourner les attributs qui ne sont pas autorisés, par exemple à partir d'une projection d'index.

- Si l'utilisateur spécifie le paramètre `ReturnValues`, sa valeur doit être `NONE`, `UPDATED_OLD` ou `UPDATED_NEW`. Cette action est obligatoire, car l'action `UpdateItem` effectue également des opérations de lecture implicites pour vérifier si un élément existe avant de le remplacer, et afin que les valeurs d'attribut précédentes puissent être retournées en cas de demande. Une telle restriction de `ReturnValues` garantit que les utilisateurs puissent uniquement lire ou écrire les attributs autorisés.
- La clause `StringEqualsIfExists` garantit qu'un seul de ces paramètres, `Select` ou `ReturnValues`, peut être utilisé par demande dans le contexte des actions autorisées.

Voici quelques variations sur cette politique :

- Pour autoriser uniquement les actions en lecture, vous pouvez supprimer `UpdateItem` de la liste des actions autorisées. Comme aucune des actions restantes n'accepte `ReturnValues`, vous pouvez supprimer `ReturnValues` de la condition. Vous pouvez également modifier `StringEqualsIfExists` en `StringEquals`, car le paramètre `Select` a toujours une valeur (`ALL_ATTRIBUTES`, sauf mention contraire).
- Pour autoriser les actions d'écriture uniquement, vous pouvez supprimer tout sauf `UpdateItem` dans la liste des actions autorisées. Comme `UpdateItem` n'utilise pas le paramètre `Select`, vous pouvez supprimer `Select` de la condition. Vous pouvez également modifier `StringEqualsIfExists` en `StringEquals`, car le paramètre `ReturnValues` a toujours une valeur (`NONE`, sauf mention contraire).
- Pour autoriser tous les attributs dont le nom correspond à un modèle, utilisez `StringLike` au lieu de `StringEquals` et utilisez un caractère générique de correspondance du modèle multi-caractère (*).

3 : accorder les autorisations pour empêcher les mises à jour de certains attributs

La politique d'autorisation suivante limite l'accès utilisateur à la seule mise à jour des attributs spécifiques identifiés par la clé de condition `dynamodb:Attributes`. La condition `StringNotLike` empêche qu'une application ne mette à jour les attributs spécifiés à l'aide de la clé de condition `dynamodb:Attributes`.

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Notez ce qui suit :

- L'action `UpdateItem`, comme les autres actions d'écriture, nécessite un accès en lecture aux éléments de façon à pouvoir retourner les valeurs avant et après la mise à jour. Dans la politique, vous limitez l'action à n'accéder qu'aux attributs qui sont autorisés à être mis à jour en spécifiant la clé de condition `dynamodb:ReturnValues`. La clé de condition limite `ReturnValues` dans la demande pour spécifier uniquement `NONE`, `UPDATED_OLD` ou `UPDATED_NEW` et n'inclut pas `ALL_OLD` ou `ALL_NEW`.
- Les actions `PutItem` et `DeleteItem` remplacent un ensemble complet, ce qui permet aux applications de modifier n'importe quel attribut. Ainsi, lorsque vous limitez une application à la mise à jour d'attributs spécifiques uniquement, vous ne devez pas accorder d'autorisation pour ceux-ci APIs.

4 : accorder les autorisations de n'interroger que les attributs projetés d'un index

La politique d'autorisations suivante autorise les requêtes sur un index secondaire (`TopScoreDateTimeIndex`) à l'aide de la clé de condition `dynamodb:Attributes`. La politique limite aussi les requêtes à ne demander que les attributs spécifiques qui ont été projetés sur l'index.

Pour demander que l'application spécifie une liste d'attributs dans la requête, la politique spécifie également la clé de condition `dynamodb:Select` pour exiger que le paramètre `Select` de l'action DynamoDB `Query` soit `SPECIFIC_ATTRIBUTES`. La liste des attributs est limitée à une liste spécifique qui est fournie à l'aide de la clé de condition `dynamodb:Attributes`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
```

```
}
```

La politique d'autorisation suivante est similaire, mais la requête doit demander tous les attributs qui ont été projetés sur l'index.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

5 : accorder les autorisations de limiter l'accès à certains attributs et valeurs de clé de partition

La politique d'autorisation suivante autorise des actions spécifiques de DynamoDB (spécifiées dans l'élément `Action`) sur une table et un index de table (spécifiés dans l'élément `Resource`). La politique utilise la clé de condition `dynamodb:LeadingKeys` pour limiter les autorisations aux seuls éléments dont la valeur de clé de partition correspond à l'ID Facebook de l'utilisateur.

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Notez ce qui suit :

- Les actions d'écriture autorisées par la politique (`UpdateItem`) peuvent uniquement modifier `attribute-A` ou `attribute-B`.
- Comme la politique autorise `UpdateItem`, une application peut insérer de nouveaux éléments et les attributs cachés ont la valeur null dans les nouveaux éléments. Si ces attributs sont projetés dans `TopScoreDateTimeIndex`, la politique présente l'avantage supplémentaire d'empêcher les requêtes qui génèrent des extractions de la table.
- Les applications ne peuvent pas lire d'autres attributs que ceux listés dans `dynamodb:Attributes`. Avec cette politique en place, une application doit définir le paramètre `Select` sur `SPECIFIC_ATTRIBUTES` dans les demandes de lecture, et seuls des attributs figurant dans la liste verte peuvent être demandés. Pour les demandes d'écriture, l'application ne peut pas définir `ReturnValues` sur `ALL_OLD` ou `ALL_NEW`, et elle ne peut pas effectuer d'opérations d'écriture conditionnelle basées sur d'autres attributs.

Rubriques en relation

- Gestion des identités et des accès pour Amazon DynamoDB
- Autorisations d'API DynamoDB : Référence des actions, ressources et conditions

Utilisation de la fédération d'identité web

Si vous écrivez une application ciblée pour un grand nombre d'utilisateurs, vous pouvez, le cas échéant, utiliser la fédération d'identité web pour l'authentification et l'autorisation. La fédération d'identité web élimine le besoin de créer des utilisateurs individuels. Au lieu de cela, les utilisateurs peuvent se connecter à un fournisseur d'identité, puis obtenir des informations d'identification de sécurité temporaires auprès de AWS Security Token Service(AWS STS). L'application peut ensuite utiliser ces informations d'identification pour accéder aux AWS services.

La fédération d'identité Web prend en charge les fournisseurs d'identité suivants :

- Login with Amazon
- Facebook
- Google

Ressources supplémentaires pour la fédération d'identité web

Les ressources suivantes peuvent vous aider à en savoir plus sur la fédération d'identité web :

- La publication Web Identity Federation using the AWS SDK pour .NET sur le blog des développeurs AWS explique comment utiliser une fédération d'identité web avec Facebook. Il inclut des extraits de code en C# qui montrent comment assumer un rôle IAM avec une identité Web et comment utiliser des informations d'identification de sécurité temporaires pour accéder à une ressource.AWS
- L'Applications mobiles AWS SDK for iOS et l'AWS Mobile SDK pour Android contiennent des exemples d'applications. Ces dernières incluent le code qui montre comment appeler les fournisseurs d'identité et comment utiliser les informations qu'ils fournissent pour obtenir et utiliser les informations d'identification de sécurité temporaires.
- L'article Fédération d'identité Web avec applications mobiles traite de la fédération d'identité Web et montre un exemple d'utilisation de la fédération d'identité Web pour accéder à une AWS ressource.

Exemple de politique pour la fédération d'identité web

Pour montrer comment utiliser la fédération d'identité Web avec DynamoDB, consultez GameScoresle tableau introduit dans. Utilisation de conditions de politique IAM pour un contrôle d'accès précis Voici la clé primaire pour GameScores.

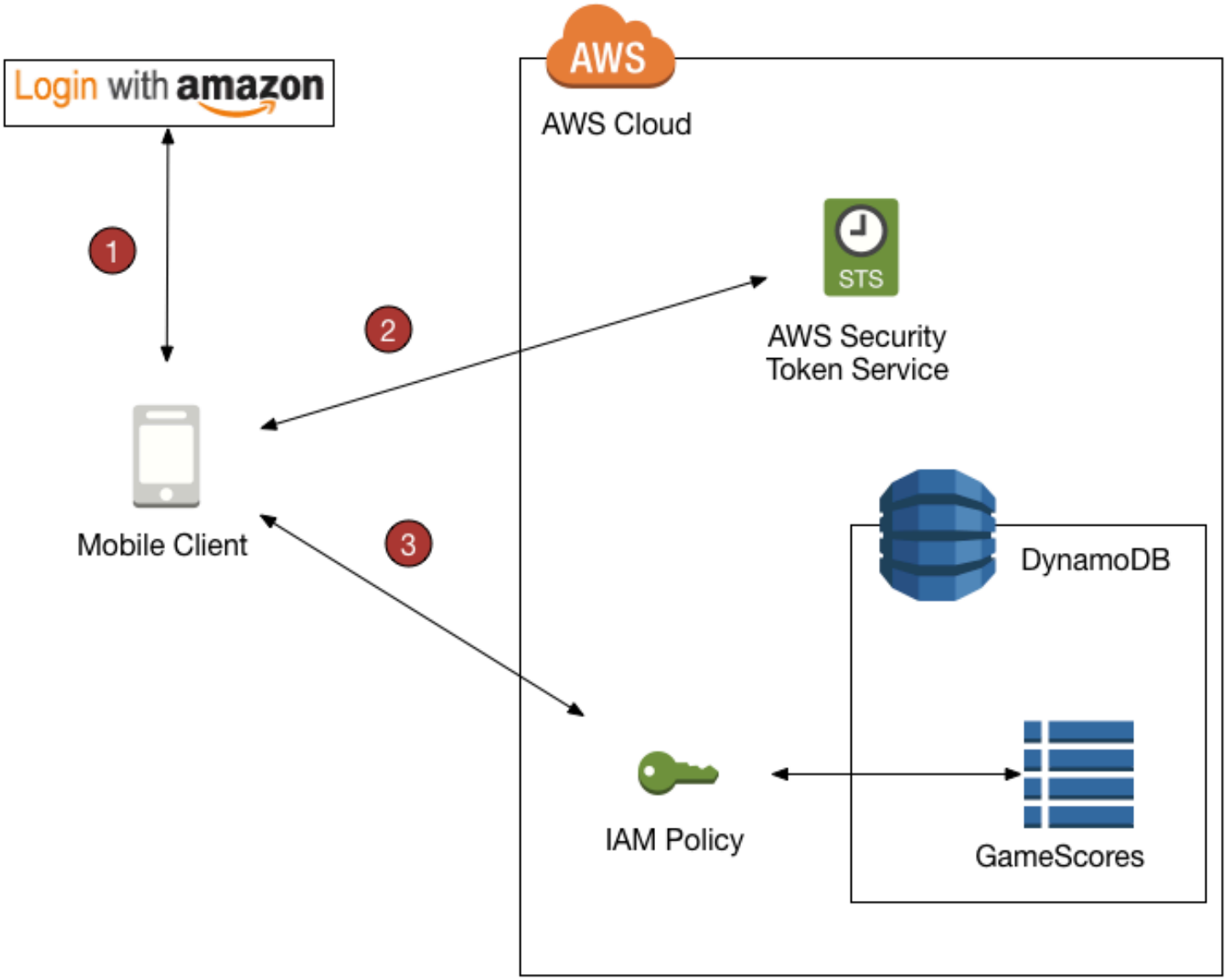
Nom de la table	Type de clé primaire	Type et nom de la clé de partition	Type et nom de la clé de tri
GameScores (UserId, GameTitle, ...)	Composite	Nom de l'attribut : UserId Type : Chaîne	Nom de l'attribut : GameTitle Type : Chaîne

Supposons maintenant qu'une application de jeu pour appareils mobiles utilise cette table et qu'elle doive prendre en charge des milliers, voire des millions, d'utilisateurs. À cette échelle, il devient très difficile de gérer les utilisateurs individuels des applications et de garantir que chaque utilisateur ne peut accéder qu'à ses propres données dans le GameScorestableau. Heureusement, comme la plupart des utilisateurs possèdent des comptes avec un fournisseur d'identité tiers, tel que Facebook, Google ou Login with Amazon. Il est donc logique de tirer parti de l'un de ces fournisseurs pour les tâches d'authentification.

Pour procéder ainsi avec la fédération d’identité web, le développeur d’application doit inscrire l’application auprès d’un fournisseur d’identité (Login with Amazon, par exemple) et obtenir un ID d’application unique. Ensuite, le développeur doit créer un rôle IAM. (Dans cet exemple, ce rôle est nommé GameRole.) Un document de politique IAM doit être joint au rôle, spécifiant les conditions dans lesquelles l'application peut accéder à la GameScorestable.

Lorsqu’un utilisateur veut jouer à un jeu, il se connecte à son compte Login with Amazon depuis l’application de jeu. L'application appelle ensuite AWS Security Token Service(AWS STS), fournit l'identifiant de l'application Login with Amazon et demande d'adhésion GameRole.AWS STS renvoie des AWS informations d'identification temporaires à l'application et lui permet d'accéder au GameScorestableau, sous réserve du document GameRolede politique.

Le schéma suivant illustre comment ces éléments s’ajustent les uns aux autres.



Présentation de la fédération d'identité web

1. L'application appelle un fournisseur d'identité tiers pour authentifier l'utilisateur et l'application. Le fournisseur d'identité retourne un jeton d'identité web à l'application.
2. L'application appelle AWS STS et transmet le jeton d'identité Web en entrée.AWS STS autorise l'application et lui donne des identifiants d'AWS accès temporaires. L'application est autorisée à assumer un rôle IAM (GameRole) et à accéder aux AWS ressources conformément à la politique de sécurité du rôle.

3. L'application appelle DynamoDB pour accéder à la table. GameScores Parce qu'elle a assumé le GameRole, l'application est soumise à la politique de sécurité associée à ce rôle. Le document de politique empêche l'application d'accéder aux données qui n'appartiennent pas à l'utilisateur.

Encore une fois, voici la politique de sécurité GameRolequi a été présentée dans Utilisation de conditions de politique IAM pour un contrôle d'accès précis :

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
```

```
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

La `Condition` clause détermine quels éléments GameScoressont visibles par l'application. A cette fin, elle compare l'ID de Login with Amazon aux valeurs de clé de partition `UserId` dans `GameScores`. Seuls les éléments appartenant à l'utilisateur actuel peuvent être traités à l'aide de l'une des actions DynamoDB répertoriées dans cette politique. Il n'est pas possible d'accéder aux autres éléments de la table. En outre, seuls les attributs spécifiques répertoriés dans la politique sont accessibles.

Préparation de l'utilisation de la fédération d'identité web

Si vous êtes un développeur d'applications et que vous souhaitez utiliser la fédération d'identité web pour votre application, procédez comme suit :

1. Inscrivez-vous en tant que développeur avec un fournisseur d'identité tiers. Les liens externes suivants fournissent des informations sur l'inscription avec les fournisseurs d'identité pris en charge :
   - Login with Amazon Developer Center
   - Inscription sur le site de Facebook
   - Utilisation de la OAuth version 2.0 pour accéder à Google APIs sur le site de Google
2. Inscrivez votre application auprès du fournisseur d'identité. Lorsque vous procédez ainsi, le fournisseur vous fournit un ID unique pour votre application. Si vous voulez que votre application fonctionne avec plusieurs fournisseurs d'identité, vous devez obtenir un ID d'application de chaque fournisseur.
3. Créez un ou plusieurs rôles IAM.Vous avez besoin d'un rôle pour chaque fournisseur d'identité de chaque application. Par exemple, vous pouvez créer un rôle qui peut être assumé par une application où l'utilisateur s'est connecté à l'aide de Login with Amazon, un deuxième rôle pour une même application où l'utilisateur s'est connecté à l'aide de Facebook et un troisième rôle pour l'application où les utilisateurs se connectent à l'aide de Google.

Dans le cadre du processus de création de rôle, vous devez attacher une politique IAM au rôle. Votre document de politique doit définir les ressources DynamoDB que votre application requiert, ainsi que les autorisations d'accès à ces ressources.

Pour plus d'informations, consultez À propos de la fédération d'identité web dans le Guide de l'utilisateur IAM.

> **Note**
>
> Au lieu de cela AWS Security Token Service, vous pouvez utiliser Amazon Cognito. Amazon Cognito est le service préféré pour la gestion des informations d'identification temporaires pour les applications mobiles. Pour plus d'informations, consultez Obtention des informations d'identification dans le Guide du développeur Amazon Cognito.

Génération d'une politique IAM à l'aide de la console DynamoDB

La console DynamoDB peut vous aider à créer une politique IAM à utiliser avec une fédération d'identité web. Pour ce faire, vous choisissez une table DynamoDB et spécifiez le fournisseur d'identité, les actions et les attributs à inclure dans la politique. La console DynamoDB génère ensuite une politique que vous pouvez attacher à un rôle IAM.

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le volet de navigation, choisissez Tables.
3. Dans la liste des tables, sélectionnez la table pour laquelle vous souhaitez créer la politique IAM.
4. Sélectionnez le bouton Actions, puis choisissez Créer une stratégie de contrôle d'accès.
5. Sélectionnez le fournisseur d'identité, les actions et les attributs de la politique.

   Lorsque les paramètres sont définis comme vous le souhaitez, choisissez Générer une stratégie. La politique générée s'affiche.
6. Choisissez Consulter la documentation, puis suivez les étapes nécessaires pour attacher la stratégie générée à un rôle IAM.

Écriture de votre application pour utiliser la fédération d’identité web

Pour utiliser une fédération d’identité web, votre application doit endosser le rôle IAM que vous avez créé. À partir de là, les applications satisfont la politique d’accès attachée au rôle.

Lors de l’exécution, si votre application utilise la fédération d’identité web, elle doit suivre cette procédure :

1. S’authentifier auprès d’un fournisseur d’identité tiers. Votre application doit appeler le fournisseur d’identité à l’aide d’une interface fournie. La façon exacte dont vous authentifiez l’utilisateur dépend du fournisseur et de la plateforme sur laquelle votre application s’exécute. Généralement, si l’utilisateur n’est pas déjà connecté, le fournisseur d’identité veille à afficher une page de connexion pour ce fournisseur.

   Une fois que le fournisseur d’identité a authentifié l’utilisateur, le fournisseur renvoie un jeton d’identité web à votre application. Le format de ce jeton dépend du fournisseur, mais il s’agit généralement d’une très longue chaîne de caractères.
2. Obtenez des informations d'identification AWS de sécurité temporaires. Pour ce faire, votre application envoie une demande `AssumeRoleWithWebIdentity` àAWS Security Token Service(AWS STS). Cette demande contient les éléments suivants :
   - Le jeton d’identité web de l’étape précédente
   - L’ID d’application du fournisseur d’identité
   - L’ARN (Amazon Resource Name) du rôle IAM que vous avez créé pour ce fournisseur d’identité pour cette application

   AWS STS renvoie un ensemble d'informations d'identification de AWS sécurité qui expirent après un certain temps (3 600 secondes, par défaut).

   Voici un exemple de demande et de réponse d’une action `AssumeRoleWithWebIdentity` dans AWS STS. Le jeton d’identité web a été obtenu depuis le fournisseur d’identité Login with Amazon.

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
```

```
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. Accédez aux AWS ressources. La réponse de AWS STS contient les informations que votre application requiert pour accéder aux ressources DynamoDB :

   - Les champs `AccessKeyID`, `SecretAccessKey` et `SessionToken` contiennent les informations d'identification de sécurité qui sont valides pour cet utilisateur et cette application uniquement.
   - Le champ `Expiration` indique le délai des informations d'identification, au terme duquel elles ne sont plus valides.
   - Le champ `AssumedRoleId` contient le nom d'un rôle IAM spécifique de la session qui a été endossé par l'application. L'application respecte les contrôles d'accès décrits dans document de politique IAM pendant la durée de la session.
   - Le champ `SubjectFromWebIdentityToken` contient l'ID unique qui s'affiche dans une variable de politique IAM pour ce fournisseur d'identité particulier. Voici les variables de politique

IAM pour les fournisseurs pris en charge, ainsi que quelques exemples de valeurs associées à ceux-ci :

Variable de politique	Exemple de valeur
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`12345678901234567890 1`

Pour des exemples de politiques IAM où ces variables de politique sont utilisées, consultez Exemples de politique : utilisation de conditions pour un contrôle d'accès détaillé.

Pour plus d'informations sur le mode de génération des AWS STS informations d'identification d'accès temporaires, consultez la section Demande d'informations d'identification de sécurité temporaires dans le guide de l'utilisateur IAM.

Autorisations d'API DynamoDB : Référence des actions, ressources et conditions

Lorsque vous configurez Gestion des identités et des accès pour Amazon DynamoDB et écrivez une politique d'autorisations que vous pouvez attacher à une identité IAM (politiques basées sur une identité), vous pouvez vous référer à la liste Actions, resources, et clés de condition pour Amazon DynamoDB dans le Guide de l'utilisateur IAM. La page répertorie chaque opération de l'API DynamoDB, les actions correspondantes pour lesquelles vous pouvez accorder des autorisations pour effectuer l'action et AWS la ressource pour laquelle vous pouvez accorder les autorisations. Vous spécifiez les actions dans le champ `Action` de la politique ainsi que la valeur des ressources dans le champ `Resource` de la politique.

Vous pouvez utiliser des AWS clés de condition larges dans vos politiques DynamoDB pour exprimer des conditions. Pour obtenir la liste complète des clés AWS-wide, consultez la référence aux éléments de politique IAM JSON dans le guide de l'utilisateur IAM.

Outre les clés de AWS condition générales, DynamoDB possède ses propres clés spécifiques que vous pouvez utiliser dans des conditions. Pour de plus amples informations, veuillez consulter Utilisation de conditions de politique IAM pour un contrôle d'accès précis.

Rubriques en relation

- Gestion des identités et des accès pour Amazon DynamoDB
- Utilisation de conditions de politique IAM pour un contrôle d'accès précis

Validation de la conformité par l'industrie pour DynamoDB

Pour savoir si un programme Services AWS de conformitéService AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Résilience et reprise après sinistre dans Amazon DynamoDB

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité.AWS Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Si vous avez besoin de répliquer vos données ou applications sur des distances géographiques plus importantes, utilisez des régions locales AWS. Une région AWS locale est un centre de données unique conçu pour compléter une AWS région existante. Comme toutes les AWS régions, les régions AWS locales sont complètement isolées des autres AWS régions.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS globale.

Outre l'infrastructure AWS mondiale, Amazon DynamoDB propose plusieurs fonctionnalités pour répondre à vos besoins en matière de résilience et de sauvegarde des données.

Sauvegarde et restauration à la demande

DynamoDB fournit une capacité de sauvegarde à la demande. Il vous permet de créer des sauvegardes complètes de vos tables pour l'archivage et la conservation à long terme. Pour plus d'informations, consultez Sauvegarde et restauration à la demande pour DynamoDB.

Point-in-time rétablissement

Point-in-time La restauration permet de protéger vos tables DynamoDB contre les opérations d'écriture ou de suppression accidentelles. Grâce à la restauration à un instant dans le passé, vous n'avez plus à vous soucier de la création, de la maintenance ou de la planification des sauvegardes à la demande. Pour plus d'informations, consultez la section Point-in-time Restauration pour DynamoDB.

Tables globales synchronisées entre les régions AWS

DynamoDB répartit automatiquement les données et le trafic de vos tables sur un nombre suffisant de serveurs afin de gérer les exigences de débit et de stockage, tout en assurant la cohérence et la rapidité des performances. Toutes vos données sont stockées sur des disques SSD (SSDs) et sont automatiquement répliquées dans plusieurs zones de disponibilité d'une AWS région, offrant ainsi une haute disponibilité et une durabilité des données intégrées. Vous pouvez utiliser des tables globales pour synchroniser les tables DynamoDB entre les régions.AWS

Sécurité de l'infrastructure dans Amazon DynamoDB

En tant que service géré, Amazon DynamoDB est protégé par les procédures de sécurité du réseau mondial décrites dans AWS la section Protection de l'infrastructure du Well-Architected Framework.AWS

Vous utilisez des appels d'API AWS publiés pour accéder à DynamoDB via le réseau. Les clients peuvent utiliser la version 1.2 ou 1.3 du protocole TLS (Transport Layer Security). Les clients doivent également prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes. En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète

associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Vous pouvez également utiliser un point de terminaison de cloud privé virtuel (VPC) pour DynamoDB afin de permettre aux EC2 instances Amazon de votre VPC d'utiliser leurs adresses IP privées pour accéder à DynamoDB sans accès à Internet public. Pour de plus amples informations, veuillez consulter Utilisation de points de terminaison d'un VPC Amazon pour accéder à DynamoDB.

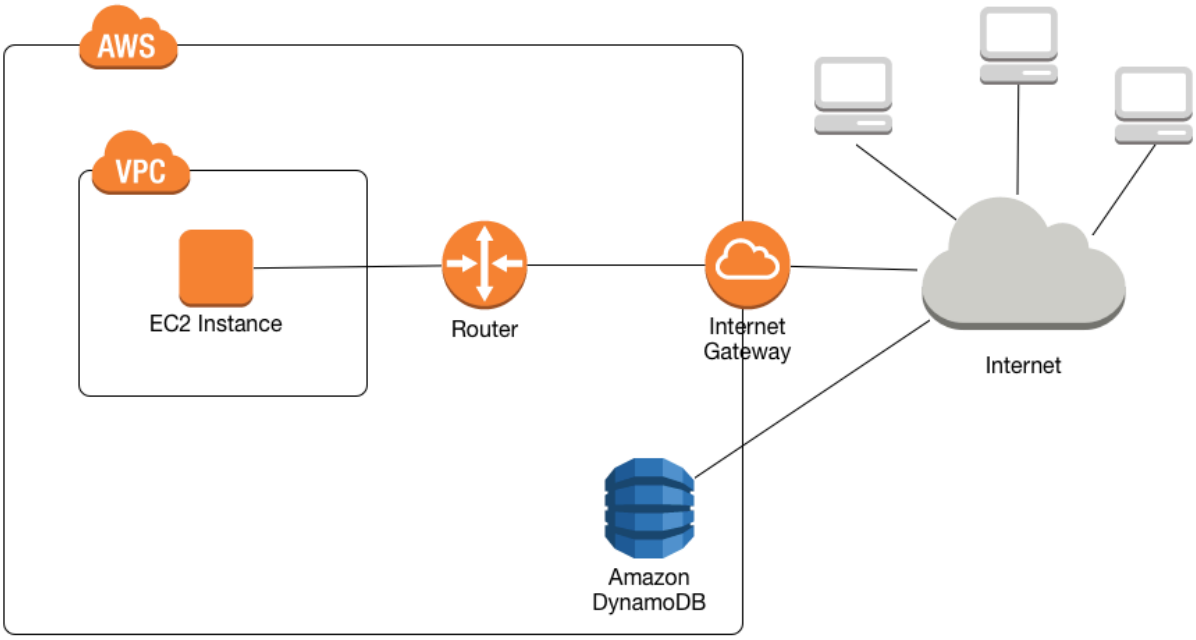
Utilisation de points de terminaison d'un VPC Amazon pour accéder à DynamoDB

Pour des raisons de sécurité, de nombreux AWS clients exécutent leurs applications dans un environnement Amazon Virtual Private Cloud (Amazon VPC). Avec Amazon VPC, vous pouvez lancer des EC2 instances Amazon dans un cloud privé virtuel, qui est logiquement isolé des autres réseaux, y compris de l'Internet public. Avec un VPC Amazon, vous contrôlez sa plage d'adresses IP, ses sous-réseaux, ses tables de routage, ses passerelles réseau et ses paramètres de sécurité.

> **Note**
>
> Si vous avez créé votre VPC Compte AWS après le 4 décembre 2013, vous avez déjà un VPC par défaut dans chacun d'eux.Région AWS Un VPC par défaut est prêt à l'emploi. Vous pouvez commencer à l'utiliser immédiatement sans avoir à le configurer.
> Pour plus d'informations, consultez VPC par défaut et sous-réseaux par défaut dans le Guide de l'utilisateur Amazon VPC.

Pour accéder à l'Internet public, votre VPC doit disposer d'une passerelle Internet, c'est-à-dire un routeur virtuel qui connecte votre VPC à Internet. Cela permet aux applications exécutées sur Amazon EC2 dans votre VPC d'accéder à des ressources Internet, telles qu'Amazon DynamoDB.

Par défaut, les communications à destination et en provenance de DynamoDB utilisent le protocole HTTPS, qui protège le trafic réseau en utilisant le chiffrement. SSL/TLS Le schéma suivant montre une EC2 instance Amazon d'un VPC accédant à DynamoDB, en faisant en sorte que DynamoDB utilise une passerelle Internet plutôt que des points de terminaison VPC.



De nombreux clients ont des préoccupations légitimes liées à la confidentialité et la sécurité lors de l'envoi et de la réception de données via le réseau Internet public. Vous pouvez balayer ces préoccupations en utilisant un réseau privé virtuel (virtual private network, VPN) pour acheminer tout le trafic réseau de DynamoDB via votre propre infrastructure réseau d'entreprise. Toutefois, cette approche peut engendrer des problèmes en matière de bande passante et de disponibilité.

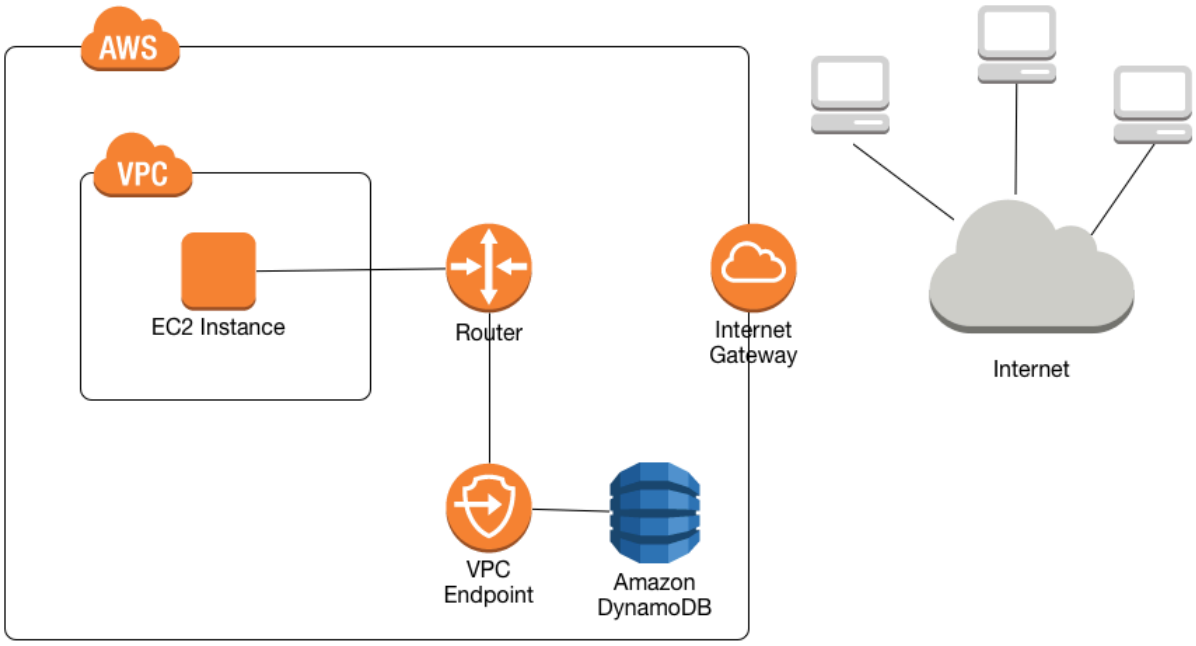
Des points de terminaison de VPC pour DynamoDB peuvent atténuer ces problèmes. Un point de terminaison VPC pour DynamoDB permet aux instances EC2 Amazon de votre VPC d'utiliser leurs adresses IP privées pour accéder à DynamoDB sans accès à Internet public. Vos EC2 instances n'ont pas besoin d'adresses IP publiques et vous n'avez pas besoin d'une passerelle Internet, d'un périphérique NAT ou d'une passerelle privée virtuelle dans votre VPC. Vous utilisez des politiques de point de terminaison pour contrôler l'accès à DynamoDB. Le trafic entre votre VPC et le AWS service ne quitte pas le réseau Amazon.

> **Note**
>
> Même lorsque vous utilisez des adresses IP publiques, toutes les communications VPC entre les instances et les services hébergés restent privées au sein du AWS réseau.AWS Les paquets provenant du AWS réseau avec une destination sur le réseau restent sur le

AWS réseau AWS mondial, à l'exception du trafic à destination ou AWS en provenance des régions de Chine.

Lorsque vous créez un point de terminaison de VPC pour DynamoDB, toutes les demandes envoyées à un point de terminaison DynamoDB au sein de la région (par exemple, dynamodb.us-west-2.amazonaws.com) sont acheminées vers un point de terminaison DynamoDB privé au sein du réseau Amazon. Vous n'avez pas besoin de modifier vos applications exécutées sur les EC2 instances de votre VPC. Le nom du point de terminaison reste le même, mais la route vers DynamoDB reste entièrement dans le réseau Amazon et n'accède pas au réseau Internet public.

Le schéma suivant montre comment une EC2 instance d'un VPC peut utiliser un point de terminaison VPC pour accéder à DynamoDB.



Pour de plus amples informations, veuillez consulter the section called “Didacticiel : Utilisation d’un point de terminaison d’un VPC pour DynamoDB”.

Partage des points de terminaison Amazon VPC et de DynamoDB

Pour permettre l'accès au service DynamoDB via le point de terminaison de passerelle d'un sous-réseau VPC, vous devez disposer des autorisations de compte propriétaire pour ce sous-réseau VPC.

Une fois que le point de terminaison de passerelle du sous-réseau VPC a obtenu l'accès à DynamoDB, n'importe quel compte AWS ayant accès à ce sous-réseau peut utiliser DynamoDB. Cela signifie que tous les utilisateurs du compte au sein du sous-réseau VPC peuvent utiliser toutes les tables DynamoDB auxquelles ils ont accès. Cela inclut les tables DynamoDB associées à un compte différent de celui du sous-réseau VPC. Le propriétaire du sous-réseau VPC peut toujours empêcher un utilisateur particulier du sous-réseau d'utiliser le service DynamoDB via le point de terminaison de passerelle, à sa discrétion.

Didacticiel : Utilisation d'un point de terminaison d'un VPC pour DynamoDB

Cette section vous guide dans la configuration et l'utilisation d'un point de terminaison de VPC pour DynamoDB.

Rubriques

- Étape 1 : Lancer une EC2 instance Amazon
- Étape 2 : configurer votre EC2 instance Amazon
- Étape 3 : créer un point de terminaison d'un VPC pour DynamoDB
- Étape 4 : (Facultatif) nettoyer

Étape 1 : Lancer une EC2 instance Amazon

Au cours de cette étape, vous lancez une EC2 instance Amazon dans votre Amazon VPC par défaut. Vous pouvez ensuite créer et utiliser un point de terminaison de VPC pour DynamoDB.

1. Ouvrez la EC2 console Amazon à l'adresse https://console.aws.amazon.com/ec2/.
2. Choisissez Lancer une instance , puis procédez comme suit :

   Étape 1 : Sélection d'une Amazon Machine Image (AMI)

   - En haut de la liste AMIs, accédez à Amazon Linux AMI et choisissez Select.

   Étape 2 : Choisir un type d'instance

- En haut de la liste des types d'instance, choisissez t2.micro.
- Sélectionnez Next: Configure Instance Details.

Étape 3 : Configurer les détails de l'instance

- Accédez à Network et choisissez votre VPC par défaut.

  Choisissez Next: Add Storage (Suivant : Ajouter le stockage).

Étape 4 : Ajouter du stockage

- Ignorez cette étape en choisissant Next: Tag Instance.

Étape 5 : étiqueter l'instance

- Ignorez cette étape en choisissant Next: Configure Security Group.

Étape 6 : Configurer un groupe de sécurité

- Choisissez Select an existing security group.
- Dans la liste des groupes de sécurité, choisissez default. Il s'agit du groupe de sécurité par défaut pour votre VPC.
- Choisissez Next: Review and Launch.

Étape 7 : Examiner le lancement de l'instance

- Choisissez Lancer.

3. Dans la fenêtre Select an existing key pair or create a new key pair, sélectionnez l'une des options suivantes :

   - Si vous ne possédez pas de paire de EC2 clés Amazon, choisissez Create a new key pair et suivez les instructions. Il vous sera demandé de télécharger un fichier de clé privée (fichier .pem) ; vous aurez besoin de ce fichier ultérieurement lorsque vous vous connecterez à votre instance Amazon EC2.

- Si vous possédez déjà une paire de EC2 clés Amazon, rendez-vous sur Sélectionner une paire de clés et choisissez votre paire de clés dans la liste. Vous devez déjà disposer du fichier de clé privée (fichier .pem) pour pouvoir vous connecter à votre instance Amazon EC2 .

4. Lorsque vous aurez configuré votre paire de clés, choisissez Launch Instances.
5. Retournez sur la page d'accueil de la EC2 console Amazon et choisissez l'instance que vous avez lancée. Dans le volet inférieur, sous l'onglet Description, recherchez le DNS public de votre instance. Par exemple : `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Notez le nom de ce DNS public, dont vous aurez besoin à l'étape suivante de ce tutoriel (Étape 2 : configurer votre EC2 instance Amazon).

> **Note**
>
> Il faudra quelques minutes pour que votre EC2 instance Amazon soit disponible. Avant de passer à l'étape suivante, assurez-vous que l'Instance State (État de l'instance) est `running` et que tous ses Status Checks (Contrôles d'état) ont réussi.

Étape 2 : configurer votre EC2 instance Amazon

Lorsque votre EC2 instance Amazon sera disponible, vous pourrez vous y connecter et la préparer pour la première utilisation.

> **Note**
>
> Les étapes suivantes supposent que vous vous connectez à votre EC2 instance Amazon depuis un ordinateur exécutant Linux. Pour d'autres méthodes de connexion, consultez Connect to Your Linux Instance dans le guide de EC2 l'utilisateur Amazon.

1. Vous devez autoriser le trafic SSH entrant vers votre instance Amazon EC2 . Pour ce faire, vous allez créer un nouveau groupe de EC2 sécurité, puis l'attribuer à votre EC2 instance.

   a. Dans le panneau de navigation, choisissez Groupes de sécurité.

   b. Sélectionnez Créer un groupe de sécurité. Dans la fenêtre Créer un groupe de sécurité, procédez comme suit :

- Security group name (Nom du groupe de sécurité) – Entrez un nom unique pour votre groupe de sécurité. Par exemple : `my-ssh-access`
- Description – Entrez une brève description du groupe de sécurité.
- VPC – Choisissez votre eVPC par défaut.
- Dans la section Security group rules (Règles du groupe de sécurité), choisissez Add Rule (Ajouter une règle), puis procédez comme suit :
  - Type – Choisissez SSH.
  - Source – Choisissez Mon IP.

Lorsque les paramètres vous conviennent, choisissez Create.

c. Dans le panneau de navigation, choisissez Instances.

d. Choisissez l' EC2 instance Amazon dans laquelle vous avez été lancéeÉtape 1 : Lancer une EC2 instance Amazon.

e. Sélectionnez Actions, Networking (Mise en réseau), Change Security Groups (Changer les groupes de sécurité).

f. Dans Change Security Groups (Modifier les groupes de sécurité), sélectionnez le groupe de sécurité que vous avez créé précédemment dans cette procédure (par exemple, `my-ssh-access`). Le groupe de sécurité `default` existant doit également être sélectionné. Lorsque les paramètres vous conviennent, choisissez Assign Security Groups (Affecter les groupes de sécurité).

2. Utilisez la `ssh` commande pour vous connecter à votre EC2 instance Amazon, comme dans l'exemple suivant.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Vous devez spécifier votre fichier de clé privée (fichier .perm) et le nom DNS public de votre instance. (Consultez Étape 1 : Lancer une EC2 instance Amazon).

L'ID de connexion est `ec2-user`. Aucun mot de passe n'est requis.

3. Configurez vos AWS informations d'identification comme indiqué dans l'exemple suivant. Saisissez vos ID de clé d'accès, clé secrète et nom de région par défaut AWS lorsque vous y êtes invité.

```
aws configure
```

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

Vous êtes maintenant prêt à créer un point de terminaison de VPC pour DynamoDB.

Étape 3 : créer un point de terminaison d'un VPC pour DynamoDB

Au cours de cette étape, vous allez créer un point de terminaison de VPC pour DynamoDB et le tester pour vérifier qu'il fonctionne.

1. Avant de commencer, vérifiez que vous pouvez communiquer avec DynamoDB à l'aide de son point de terminaison public.

   ```
   aws dynamodb list-tables
   ```

   La sortie affiche la liste des tables DynamoDB que vous possédez actuellement (si vous n'avez aucune table, la liste est vide).
2. Vérifiez que DynamoDB est un service disponible pour créer des points de terminaison VPC dans la région actuelle.AWS(la commande est affichée en gras, suivie d'un exemple de sortie).

   ```
   aws ec2 describe-vpc-endpoint-services
   ```

   ```
   {
       "ServiceNames": [
           "com.amazonaws.us-east-1.s3",
           "com.amazonaws.us-east-1.dynamodb"
       ]
   }
   ```

   Dans l'exemple de sortie, DynamoDB est l'un des services disponibles. Vous pouvez donc continuer à créer un point de terminaison de VPC pour celui-ci.
3. Déterminez votre identifiant de VPC.

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

Dans l'exemple de sortie, l'ID de VPC est `vpc-0bbc736e`.

4. Créez le point de terminaison de VPC. Pour le paramètre `--vpc-id`, spécifiez l'ID de VPC de l'étape précédente. Utilisez le paramètre `--route-table-ids` pour associer le point de terminaison à vos tables de routage.

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-
east-1.dynamodb --route-table-ids rtb-11aa22bb
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. Vérifiez que vous pouvez accéder à DynamoDB via le point de terminaison de VPC.

```
aws dynamodb list-tables
```

Si vous le souhaitez, vous pouvez essayer d'autres AWS CLI commandes pour DynamoDB. Pour plus d'informations, consultez la référence de la commande AWS CLI.

Étape 4 : (Facultatif) nettoyer

Si vous souhaitez supprimer les ressources que vous avez créées dans ce didacticiel, procédez comme suit :

Pour supprimer votre point de terminaison de VPC pour DynamoDB

1. Connectez-vous à votre EC2 instance Amazon.
2. Déterminez l'ID du point de terminaison de VPC.

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

Dans l'exemple de sortie, l'ID du point de terminaison de VPC est `vpce-9b15e2f2`.

3. Supprimez le point de terminaison de VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
```

```
}
```

Le tableau vide `[]` indique le succès de l'opération (il n'y a pas eu de demande infructueuse).

Pour mettre fin à votre EC2 instance Amazon

1. Ouvrez la EC2 console Amazon à l'adresse https://console.aws.amazon.com/ec2/.
2. Dans le panneau de navigation, choisissez Instances.
3. Choisissez votre EC2 instance Amazon.
4. Choisissez Actions, Instance State (État de l'instance), Terminate (Résilier).
5. Dans la fenêtre de confirmation, choisissez Yes, Terminate (Oui, Résilier).

AWS PrivateLink pour DynamoDB

Avec AWS PrivateLink DynamoDB, vous pouvez provisionner des points de terminaison Amazon VPC (points de terminaison d'interface) dans votre cloud privé virtuel (Amazon VPC). Ces points de terminaison sont directement accessibles depuis des applications installées sur site via un VPN et/ou via un autre système de peering Région AWS via Amazon VPC.Direct Connect En utilisant AWS PrivateLink et les points de terminaison d'interface, vous pouvez simplifier la connectivité réseau privé entre vos applications et DynamoDB.

Les applications de votre VPC n'ont pas besoin d'adresses IP publiques pour communiquer avec DynamoDB à l'aide des points de terminaison d'interface VPC pour les opérations DynamoDB. Les points de terminaison d'interface sont représentés par une ou plusieurs interfaces réseau élastiques (ENIs) auxquelles des adresses IP privées sont attribuées à partir de sous-réseaux de votre Amazon VPC. Les demandes adressées à DynamoDB via les points de terminaison d'interface restent sur le réseau Amazon. Vous pouvez également accéder aux points de terminaison d'interface de votre Amazon VPC à partir d'applications sur site AWS Direct Connect via AWS Virtual Private Network ou ().Site-to-Site VPN Pour plus d'informations sur la façon de connecter votre VPC Amazon à votre réseau sur site, consultez le Guide de l'utilisateur Direct Connect et le Guide de l'utilisateur AWS Site-to-Site VPN.

Pour des informations générales sur les points de terminaison d'interface, consultez Interface Amazon VPC endpoints AWS PrivateLink() dans le Guide.AWS PrivateLinkAWS PrivateLink est également pris en charge pour les points de terminaison Amazon DynamoDB Streams. Pour de

plus amples informations, veuillez consulter the section called “AWS PrivateLink pour DynamoDB Streams”.

Rubriques

- Types de points de terminaison d’un VPC Amazon pour Amazon DynamoDB
- Considérations relatives à l'utilisation AWS PrivateLink d'Amazon DynamoDB
- Création d’un point de terminaison Amazon VPC
- Accès aux points de terminaison de l’interface Amazon DynamoDB
- Accès aux tables DynamoDB et opérations d’API de contrôle depuis les points de terminaison de l’interface DynamoDB
- Mise à jour d’une configuration DNS sur site
- Création d’une politique de point de terminaison d’un VPC Amazon pour DynamoDB
- Utilisation de points de terminaison DynamoDB avec AWS Management Console Private Access
- AWS PrivateLink pour DynamoDB Streams
- Utilisation AWS PrivateLink pour DynamoDB Accelerator (DAX)

Types de points de terminaison d’un VPC Amazon pour Amazon DynamoDB

Vous pouvez utiliser deux types de points de terminaison Amazon VPC pour accéder à Amazon DynamoDB : les points de terminaison de passerelle et les points de terminaison d'interface (en utilisant).AWS PrivateLink Un point de terminaison de passerelle est une passerelle que vous spécifiez dans votre table de routage pour accéder à DynamoDB depuis votre Amazon VPC via le réseau.AWS Les points de terminaison d'interface étendent les fonctionnalités des points de terminaison de passerelle en utilisant des adresses IP privées pour acheminer les demandes vers DynamoDB depuis votre Amazon VPC, sur site, ou depuis un Amazon VPC dans un autre en utilisant Amazon VPC peering ou.Région AWSAWS Transit Gateway Pour plus d’informations, consultez What is Amazon VPC peering? et Transit Gateway vs Amazon VPC peering.

Les points de terminaison d’interface sont compatibles avec les points de terminaison de passerelle. Si vous avez un point de terminaison de passerelle existant dans le VPC Amazon, vous pouvez utiliser les deux types de points de terminaison dans le même VPC Amazon.

Points de terminaison de passerelle pour DynamoDB	Points de terminaison d'interface pour DynamoDB
Dans les deux cas, votre trafic réseau reste sur le AWS réseau.	
Utiliser des adresses IP publiques Amazon DynamoDB	Utiliser des adresses IP privées depuis votre VPC Amazon pour accéder à Amazon DynamoDB
N'autorise pas l'accès sur site	Autoriser l'accès depuis vos sites
Ne pas autoriser l'accès depuis un autre Région AWS	Autorisez l'accès depuis un point de terminaison Amazon VPC à un autre en Région AWS utilisant Amazon VPC peering ou AWS Transit Gateway
Non facturé	Facturé

Pour plus d'informations sur les points de terminaison de passerelle, consultez Gateway Amazon VPC endpoints dans le Guide AWS PrivateLink.

Considérations relatives à l'utilisation AWS PrivateLink d'Amazon DynamoDB

Les considérations relatives à Amazon VPC s'appliquent à Amazon AWS PrivateLink DynamoDB. Pour plus d'informations, consultez Considérations sur les points de terminaison d'interface et Quotas AWS PrivateLink dans le Guide AWS PrivateLink. En outre, les restrictions suivantes s'appliquent.

AWS PrivateLink pour Amazon DynamoDB ne prend pas en charge les éléments suivants :

- Protocole TLS (Transport Layer Security) 1.1
- Services de système de nom de domaine (DNS) privé et hybride

Vous pouvez envoyer jusqu'à 50 000 demandes par seconde pour chaque AWS PrivateLink point de terminaison que vous activez.

> **Note**
>
> Les délais de connectivité réseau vers les AWS PrivateLink points de terminaison ne sont pas concernés par les réponses d'erreur DynamoDB et doivent être gérés de manière appropriée par les applications qui se connectent aux points de terminaison. PrivateLink

Création d’un point de terminaison Amazon VPC

Pour créer un point de terminaison d’interface d’un VPC Amazon, consultez Create an Amazon VPC endpoint dans le Guide AWS PrivateLink.

Accès aux points de terminaison de l’interface Amazon DynamoDB

Lorsque vous créez un point de terminaison d’interface, DynamoDB génère deux types de noms DNS DynamoDB spécifiques au point de terminaison : des noms régionaux et des noms zonaux.

- Un nom DNS régional inclut un identifiant de point de terminaison Amazon VPC unique, un identifiant de service Région AWS, le et `vpce.amazonaws.com` dans son nom. Par exemple, pour l’ID de point de terminaison d’un VPC Amazon *`vpce-1a2b3c4d`*, le nom DNS généré peut être similaire à *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com`.
- Les noms DNS zonaux incluent la zone de disponibilité, par exemple, *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`. Vous pouvez utiliser cette option si votre architecture isole les zones de disponibilité. Par exemple, vous pouvez l’utiliser pour contenir les pannes ou réduire les coûts de transfert de données Régionaux.

> **Note**
>
> Pour obtenir une fiabilité optimale, nous vous recommandons de déployer votre service dans au moins trois zones de disponibilité.

Accès aux tables DynamoDB et opérations d’API de contrôle depuis les points de terminaison de l’interface DynamoDB

Vous pouvez utiliser le AWS CLI ou AWS SDKs pour accéder aux tables DynamoDB et contrôler les opérations de l'API via les points de terminaison de l'interface DynamoDB.

AWS CLI exemples

Pour accéder aux tables DynamoDB ou aux opérations de l'API de contrôle DynamoDB via les points de terminaison de l'interface DynamoDB dans les commandes, utilisez les paramètres et.AWS CLI`--region` `--endpoint-url`

Exemple : création du point de terminaison d'un VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Exemple : modifier le point de terminaison d'un VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Exemple : lister les tables à l'aide d'une URL de point de terminaison

Dans l'exemple suivant, remplacez la région `us-east-1` et le nom DNS de l'ID du point de terminaison d'un VPC `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS Exemples de SDK

Pour accéder aux tables DynamoDB ou aux opérations de l'API de contrôle DynamoDB via les points de terminaison de l'interface DynamoDB lorsque vous utilisez le, installez la dernière version.AWS SDKs SDKs Configurez ensuite vos clients pour qu'ils utilisent une URL de point de terminaison afin d'accéder à une table ou à une opération API de contrôle de DynamoDB via des points de terminaison d'interface DynamoDB.

SDK for Python (Boto3)

Exemple : Utiliser une URL de point de terminaison pour accéder à une table DynamoDB

Dans l'exemple suivant, remplacez la région `us-east-1` et l'ID du point de terminaison d'un VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Exemple : Utiliser une URL de point de terminaison pour accéder à une table DynamoDB

Dans l'exemple suivant, remplacez la région `us-east-1` et l'ID du point de terminaison d'un VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Exemple : Utiliser une URL de point de terminaison pour accéder à une table DynamoDB

Dans l'exemple suivant, remplacez le Region us-east-1 et l'ID de point de terminaison https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com VPC par vos propres informations.

```
Region region = Region.US_EAST_1;
```

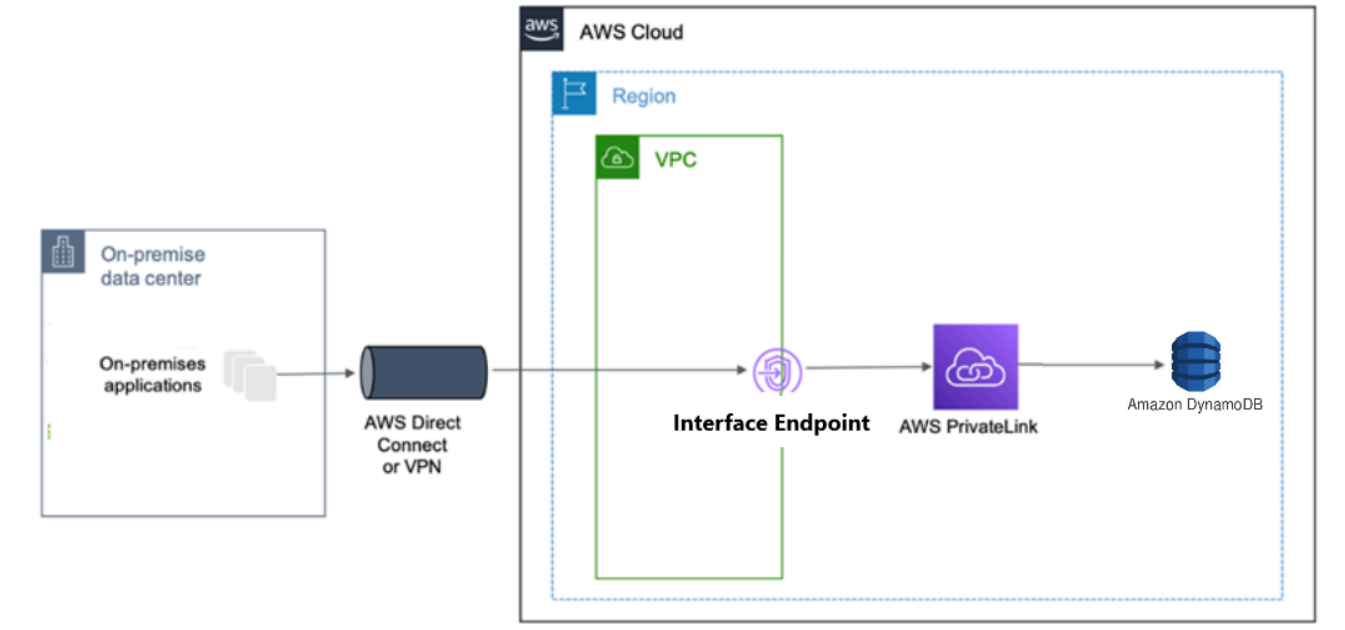
```
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Mise à jour d'une configuration DNS sur site

Lorsque vous utilisez des noms DNS spécifiques aux points de terminaison pour accéder aux points de terminaison d'interface pour DynamoDB, vous n'avez pas besoin de mettre à jour votre résolveur DNS sur site. Vous pouvez résoudre le nom DNS spécifique au point de terminaison avec l'adresse IP privée du point de terminaison d'interface depuis le domaine DNS public DynamoDB.

Utilisation de points de terminaison d'interface pour accéder à DynamoDB sans point de terminaison de passerelle ou passerelle Internet dans le VPC Amazon

Les points de terminaison d'interface de votre VPC Amazon peuvent acheminer les applications de VPC Amazon et les applications sur site vers DynamoDB via le réseau Amazon, comme illustré dans le diagramme suivant.



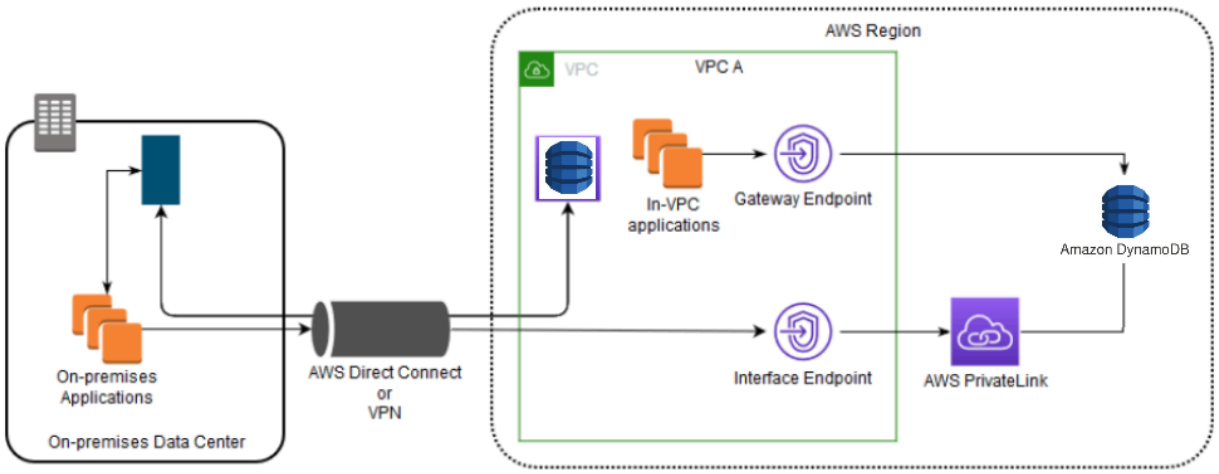
Le diagramme illustre les éléments suivants :

- Votre réseau sur site utilise Direct Connect ou Site-to-Site VPN pour se connecter à Amazon VPC A.

- Vos applications sur site et dans le VPC Amazon A utilisent des noms DNS spécifiques aux points de terminaison pour accéder à DynamoDB via le point de terminaison d'interface DynamoDB.
- Les applications sur site envoient des données au point de terminaison de l'interface dans l'Amazon VPC Direct Connect via (ou)Site-to-Site VPN.AWS PrivateLink déplace les données du point de terminaison de l'interface vers DynamoDB via le réseau.AWS
- Les applications VPC d'Amazon envoient également du trafic vers le point de terminaison de l'interface.AWS PrivateLink déplace les données du point de terminaison de l'interface vers DynamoDB via le réseau.AWS

Utilisation conjointe de points de terminaison de passerelle et de points de terminaison d'interface dans le même VPC Amazon pour accéder à DynamoDB

Vous pouvez créer des points de terminaison d'interface et conserver le point de terminaison de passerelle existant dans le même VPC Amazon, comme le montre le diagramme suivant. En adoptant cette approche, vous autorisez les applications internes au VPC Amazon à continuer d'accéder à DynamoDB via le point de terminaison de la passerelle, ce qui n'est pas facturé. Ensuite, seules vos applications sur site utilisent des points de terminaison d'interface pour accéder à DynamoDB. Pour accéder à DynamoDB de cette façon, vous devez mettre à jour vos applications sur site afin d'utiliser des noms DNS spécifiques du point de terminaison pour DynamoDB.



Le diagramme illustre les éléments suivants :

- Les applications sur site utilisent des noms DNS spécifiques au point de terminaison pour envoyer des données au point de terminaison de l'interface au sein d'Amazon VPC via (ou).Direct

ConnectSite-to-Site VPNAWS PrivateLink déplace les données du point de terminaison de l'interface vers DynamoDB via le réseau.AWS

- À l'aide des noms DynamoDB régionaux par défaut, les applications VPC intégrées à Amazon envoient des données au point de terminaison de la passerelle qui se connecte à DynamoDB via le réseau.AWS

Pour plus d'informations sur les points de terminaison de passerelle, consultez Points de terminaison de VPC Amazon de passerelle dans le Guide d'utilisateur Amazon VPC.

Création d'une politique de point de terminaison d'un VPC Amazon pour DynamoDB

Vous pouvez attacher une politique de point de terminaison à votre point de terminaison d'un VPC Amazon qui contrôle l'accès à DynamoDB. La politique spécifie les informations suivantes :

- Le principal Gestion des identités et des accès AWS(IAM) qui peut effectuer des actions
- Les actions qui peuvent être effectuées.
- Les ressources sur lesquelles les actions peuvent être exécutées.

Rubriques

- Exemple : restriction de l'accès à une table depuis le point de terminaison d'un VPC Amazon

Exemple : restriction de l'accès à une table depuis le point de terminaison d'un VPC Amazon

Vous pouvez créer une politique de point de terminaison qui restreint l'accès uniquement à des tables DynamoDB spécifiques. Ce type de politique est utile si d'autres politiques Services AWS de votre Amazon VPC utilisent des tables. La politique de table suivante restreint l'accès à *DOC-EXAMPLE-TABLE* uniquement. Pour utiliser cette politique de point de terminaison, remplacez *DOC-EXAMPLE-TABLE* par le nom de votre table.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
```

```
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
                   "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
}
```

Utilisation de points de terminaison DynamoDB avec AWS Management Console Private Access

Vous devez paramétrer la configuration DNS pour DynamoDB et DynamoDB Streams lorsque vous utilisez des points de terminaison d'un VPC avec la console DynamoDB dans AWS Management Console Private Access.

Pour configurer DynamoDB de manière à ce qu'il soit accessible AWS Management Console dans Private Access, vous devez créer les deux points de terminaison VPC suivants :

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Lorsque vous créez les points de terminaison d'un VPC, accédez à la console Route53 et créez une zone hébergée privée pour DynamoDB à l'aide du point de terminaison régional `dynamodb.us-east-1.amazonaws.com`.

Créez les deux enregistrements d'alias suivants dans la zone hébergée privée :

- `dynamodb.<region>.amazonaws.com` qui achemine le trafic vers le point de terminaison d'un VPC `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` qui achemine le trafic vers le point de terminaison d'un VPC `com.amazonaws.<region>.dynamodb-streams`.

AWS PrivateLink pour DynamoDB Streams

Avec AWS PrivateLink Amazon DynamoDB Streams, vous pouvez provisionner des points de terminaison Amazon VPC (points de terminaison d'interface) dans votre cloud privé virtuel (Amazon VPC). Ces points de terminaison sont directement accessibles depuis des applications installées sur site via un VPN et/ou via un autre système de peering Région AWS via Amazon VPC.Direct ConnectÀ l'aide de points de terminaison AWS PrivateLink et d'interface, vous pouvez simplifier la connectivité au réseau privé entre vos applications et DynamoDB Streams.

Les applications de votre VPC Amazon n'ont pas besoin d'adresses IP publiques pour communiquer avec DynamoDB Streams à l'aide des points de terminaison de l'interface Amazon VPC pour les opérations DynamoDB Streams. Les points de terminaison d'interface sont représentés par une ou plusieurs interfaces réseau élastiques (ENIs) auxquelles des adresses IP privées sont attribuées à partir de sous-réseaux de votre Amazon VPC. Les demandes adressées à DynamoDB Streams via les points de terminaison d'interface restent sur le réseau Amazon. Vous pouvez également accéder aux points de terminaison d'interface de votre Amazon VPC à partir d'applications sur site Direct Connect via AWS Virtual Private Network ou AWS(VPN). Pour plus d'informations sur la façon de vous connecter AWS Virtual Private Networkà votre réseau local, consultez le guide de l'Direct Connect utilisateur et le guide de l'AWS Site-to-Site VPN utilisateur.

Pour des informations générales sur les points de terminaison d'interface, consultez Interface de points de terminaison d'un réseau Amazon VPC (AWS PrivateLink).

> **Note**
>
> Seuls les points de terminaison d'interface sont pris en charge pour DynamoDB Streams. Les points de terminaison de passerelle ne sont pas pris en charge.

Rubriques

- Considérations relatives à l'utilisation de AWS PrivateLink pour Amazon DynamoDB Streams
- Création d'un point de terminaison Amazon VPC
- Accès aux points de terminaison de l'interface Amazon DynamoDB Streams
- Accès aux opérations d'API DynamoDB Streams depuis les points de terminaison de l'interface DynamoDB Streams
- AWS Exemples de SDK
- Création d'une politique de point de terminaison d'un VPC Amazon pour DynamoDB Streams

- Utilisation de points de terminaison DynamoDB avec AWS Management Console Private Access

Considérations relatives à l'utilisation de AWS PrivateLink pour Amazon DynamoDB Streams

Les considérations relatives à Amazon VPC s'appliquent àAWS PrivateLink Amazon DynamoDB Streams. Pour plus d'informations, consultez interface endpoint considerations et Quotas AWS PrivateLink. Les restrictions suivantes s'appliquent.

AWS PrivateLink pour Amazon DynamoDB Streams ne prend pas en charge les éléments suivants :

- Protocole TLS (Transport Layer Security) 1.1
- Services de système de nom de domaine (DNS) privé et hybride

> **Note**
>
> Les délais de connectivité réseau vers les AWS PrivateLink points de terminaison ne sont pas concernés par les réponses d'erreur de DynamoDB Streams et doivent être gérés de manière appropriée par les applications qui se connectent aux points de terminaison.AWS PrivateLink

Création d'un point de terminaison Amazon VPC

Pour créer un point de terminaison d'interface d'un VPC Amazon, consultez Create an Amazon VPC endpoint dans le Guide AWS PrivateLink.

Accès aux points de terminaison de l'interface Amazon DynamoDB Streams

Lorsque vous créez un point de terminaison d'interface, DynamoDB génère deux types de noms DNS DynamoDB Streams spécifiques au point de terminaison : des noms régionaux et des noms zonaux.

- Un nom DNS régional inclut un identifiant de point de terminaison Amazon VPC unique, un identifiant de service Région AWS, le et `vpce.amazonaws.com` dans son nom. Par exemple, pour l'ID de point de terminaison d'un VPC Amazon *`vpce-1a2b3c4d`*, le nom DNS généré peut être similaire à *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`.
- Les noms DNS zonaux incluent la zone de disponibilité, par exemple, *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.streams.dynamodb.us-`

`east-1.vpce.amazonaws.com`. Vous pouvez utiliser cette option si votre architecture isole les zones de disponibilité. Par exemple, vous pouvez l'utiliser pour contenir les pannes ou réduire les coûts de transfert de données Régionaux.

Accès aux opérations d'API DynamoDB Streams depuis les points de terminaison de l'interface DynamoDB Streams

Vous pouvez utiliser le AWS CLI ou AWS SDKs pour accéder aux opérations de l'API DynamoDB Streams via les points de terminaison de l'interface DynamoDB Streams.

AWS CLI exemples

Pour accéder aux flux DynamoDB ou aux opérations d'API via les points de terminaison de l'interface DynamoDB Streams dans les commandes, utilisez les paramètres et.AWS CLI`--region` `--endpoint-url`

Exemple : création du point de terminaison d'un VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb-streams \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Exemple : modifier le point de terminaison d'un VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Exemple : lister les flux à l'aide d'une URL de point de terminaison

Dans l'exemple suivant, remplacez la région `us-east-1` et le nom DNS de l'ID du point de terminaison d'un VPC `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

AWS Exemples de SDK

Pour accéder aux opérations de l'API Amazon DynamoDB Streams via les points de terminaison de l'interface DynamoDB Streams lorsque vous utilisez le, installez la dernière version.AWS SDKs SDKs Configurez ensuite vos clients pour qu'ils utilisent une URL de point de terminaison pour une opération d'API DynamoDB Streams via des points de terminaison d'interface DynamoDB Streams.

SDK for Python (Boto3)

Exemple : utiliser une URL de point de terminaison pour accéder à un flux DynamoDB

Dans l'exemple suivant, remplacez la région `us-east-1` et l'ID du point de terminaison d'un VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
ddb_streams_client = session.client(
service_name='dynamodbstreams',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Exemple : utiliser une URL de point de terminaison pour accéder à un flux DynamoDB

Dans l'exemple suivant, remplacez la région `us-east-1` et l'ID du point de terminaison d'un VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
//client build with endpoint config
final AmazonDynamoDBStreams dynamodbstreams =
 AmazonDynamoDBStreamsClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
```

```
        )
).build();
```

SDK for Java 2.x

Exemple : utiliser une URL de point de terminaison pour accéder à un flux DynamoDB

Dans l'exemple suivant, remplacez la région `us-east-1` et l'ID du point de terminaison d'un VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
Region region = Region.US_EAST_1;
dynamoDbStreamsClient = DynamoDbStreamsClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Création d'une politique de point de terminaison d'un VPC Amazon pour DynamoDB Streams

Vous pouvez attacher une politique de point de terminaison à votre point de terminaison Amazon VPC qui contrôle l'accès à DynamoDB Streams. La politique spécifie les informations suivantes :

- Le principal Gestion des identités et des accès AWS(IAM) qui peut effectuer des actions
- Les actions qui peuvent être effectuées.
- Les ressources sur lesquelles les actions peuvent être exécutées.

Rubriques

- Exemple : restriction de l'accès à un flux spécifique depuis le point de terminaison d'un VPC Amazon

Exemple : restriction de l'accès à un flux spécifique depuis le point de terminaison d'un VPC Amazon

Vous pouvez créer une politique de point de terminaison qui restreint l'accès uniquement à des DynamoDB Streams spécifiques. Ce type de politique est utile si d'autres Services AWS politiques de votre Amazon VPC utilisent DynamoDB Streams. La politique de diffusion suivante restreint l'accès au flux *2025-02-20T11:22:33.444* attaché à *DOC-EXAMPLE-TABLE*. Pour utiliser cette

politique de point de terminaison, remplacez *DOC-EXAMPLE-TABLE* par le nom de votre table et *2025-02-20T11:22:33.444* par l'étiquette de flux.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> Les points de terminaison de passerelle ne sont pas pris en charge dans DynamoDB Streams.

Utilisation de points de terminaison DynamoDB avec AWS Management Console Private Access

Vous devez paramétrer la configuration DNS pour DynamoDB et DynamoDB Streams lorsque vous utilisez des points de terminaison d'un VPC avec la console DynamoDB dans AWS Management Console Private Access.

Pour configurer DynamoDB de manière à ce qu'il soit accessible AWS Management Console dans Private Access, vous devez créer les deux points de terminaison VPC suivants :

- `com.amazonaws.<region>.dynamodb`

- com.amazonaws.<region>.dynamodb-streams

Lorsque vous créez les points de terminaison d'un VPC, accédez à la console Route53 et créez une zone hébergée privée pour DynamoDB à l'aide du point de terminaison régional dynamodb.us-east-1.amazonaws.com.

Créez les deux enregistrements d'alias suivants dans la zone hébergée privée :

- dynamodb.<region>.amazonaws.com qui achemine le trafic vers le point de terminaison d'un VPC com.amazonaws.<region>.dynamodb.
- streams.dynamodb.<region>.amazonaws.com qui achemine le trafic vers le point de terminaison d'un VPC com.amazonaws.<region>.dynamodb-streams.

Utilisation AWS PrivateLink pour DynamoDB Accelerator (DAX)

AWS PrivateLink pour DynamoDB Accelerator (DAX) vous permet d'accéder en toute sécurité à la APIs gestion DAX, par exemple CreateClusterDescribeClusters, DeleteCluster et via des adresses IP privées au sein de votre cloud privé virtuel (VPC). Cette fonctionnalité vous permet d'accéder aux services DAX de manière privée à partir de vos applications sans exposer le trafic à l'Internet public.

Le DAX PrivateLink prend en charge les points de terminaison à double pile (dax.{region}.api.aws), ce qui permet à la fois la connectivité IPv4 . IPv6 Avec AWS PrivateLink for DAX, les clients peuvent accéder au service en utilisant des noms DNS privés. La prise en charge des terminaux à double pile garantit une connectivité transparente tout en préservant la confidentialité du réseau. Cela vous permet d'accéder à DAX via Internet public et des points de terminaison VPC sans apporter de modifications à la configuration de votre SDK.

Considérations relatives à l'utilisation AWS PrivateLink de DynamoDB Accelerator (DAX)

Lors de l'implémentation AWS PrivateLink pour DynamoDB Accelerator (DAX), plusieurs considérations importantes doivent être prises en compte.

Avant de configurer un point de terminaison d'interface pour DAX, tenez compte des points suivants :

- Les points de terminaison de l'interface DAX ne prennent en charge que l'accès à la gestion DAX APIs au sein de celle-ci.Région AWS Vous ne pouvez pas utiliser un point de terminaison d'interface pour accéder à la gestion DAX APIs dans d'autres régions.
- Pour accéder au mode AWS Management Console privé à des fins de gestion DAX, vous devrez peut-être créer des points de terminaison VPC supplémentaires pour des services `com.amazonaws.region.console` tels que les services connexes.
- La création et l'utilisation d'un point de terminaison d'interface pour DAX vous sont facturées. Pour en savoir plus sur la tarification, consultez Tarification AWS PrivateLink.

Comment AWS PrivateLink fonctionne avec DAX

Lorsque vous créez un point de terminaison d'interface pour DAX :

1. AWS crée une interface réseau de point de terminaison dans chaque sous-réseau que vous activez pour le point de terminaison de l'interface.
2. Il s'agit d'interfaces réseau gérées par les demandeurs qui servent de points d'entrée pour le trafic destiné au DAX.
3. Vous pouvez ensuite accéder au DAX via des adresses IP privées au sein de votre VPC.
4. Cette architecture vous permet d'utiliser des groupes de sécurité VPC pour gérer l'accès aux points de terminaison.
5. Les applications peuvent accéder à DynamoDB et DAX via leurs points de terminaison d'interface respectifs au sein d'un VPC, tout en permettant aux applications locales de se connecter via Direct Connect ou VPN.
6. Cela fournit un modèle de connectivité cohérent pour les deux services, simplifie l'architecture et améliore la sécurité en maintenant le trafic au sein du AWS réseau.

Création de points de terminaison d'interface pour DAX

Vous pouvez créer un point de terminaison d'interface pour vous connecter à DAX à l'aide du AWS Management ConsoleAWS SDK ou de l'AWS API.CloudFormation

Pour créer un point de terminaison d'interface pour DAX à l'aide de la console

1. Accédez à la console Amazon VPC à l'adresse. https://console.aws.amazon.com/vpc/
2. Dans le panneau de navigation, choisissez Points de terminaison.

3. Choisissez Create Endpoint (Créer un point de terminaison).
4. Pour la catégorie de service, choisissez Services AWSet pour le nom du service, recherchez et sélectionnez`com.amazonaws.region.dax`.
5. Pour le VPC, sélectionnez le VPC à partir duquel vous souhaitez accéder au DAX et pour les sous-réseaux, sélectionnez les sous-réseaux dans lesquels AWS seront créées les interfaces réseau des points de terminaison.
6. Pour les groupes de sécurité, sélectionnez ou créez des groupes de sécurité à associer aux interfaces réseau des terminaux.
7. Pour Policy, conservez l'accès complet par défaut ou personnalisez-le selon vos besoins.
8. Sélectionnez Activer le nom DNS pour activer le DNS privé pour le point de terminaison. Maintenez le nom DNS privé activé pour empêcher toute modification de la configuration du SDK. Lorsque cette option est activée, vos applications peuvent continuer à utiliser le nom DNS du service standard (exemple :`dax.region.amazonaws.com`).AWS crée une zone hébergée privée dans votre VPC qui convertit ce nom en adresse IP privée de votre point de terminaison.

   > **Note**
   >
   > Utilisez des noms DNS régionaux si nécessaire. L'utilisation de noms DNS zonaux n'est pas recommandée. Sélectionnez également des sous-réseaux parmi 3 ou plus AZs pour garantir une disponibilité maximale. PrivateLink

9. Choisissez Créer un point de terminaison.

Pour créer un point de terminaison d'interface pour DAX à l'aide du AWS CLI

Utilisez la `create-vpc-endpoint` commande avec le `vpc-endpoint-type` paramètre défini sur `Interface` et le `service-name` paramètre défini sur`com.amazonaws.region.dax`.

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-ec43eb89 \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.us-east-1.dax \
    --subnet-ids subnet-abcd1234 subnet-1a2b3c4d \
    --security-group-ids sg-1a2b3c4d \
    --private-dns-enabled
```

Ressources supplémentaires

Pour plus d'informations sur AWS PrivateLink les points de terminaison VPC, consultez les ressources suivantes :

- AWS PrivateLink pour DynamoDB
- AWS PrivateLink pour DynamoDB Streams
- Connectez votre VPC aux services à l'aide de AWS PrivateLink
- Simplifiez la connectivité privée à DynamoDB avec AWS PrivateLink
- AWS PrivateLink Livre blanc

Configuration et analyse des vulnérabilités dans Amazon DynamoDB

AWS gère les tâches de sécurité de base telles que l'application de correctifs au système d'exploitation client (OS) et aux bases de données, la configuration du pare-feu et la reprise après sinistre. Ces procédures ont été vérifiées et certifiées par les tiers appropriés. Pour plus de détails, consultez les ressources suivantes :

- Validation de conformité pour Amazon DynamoDB
- Modèle de responsabilité partagée
- Amazon Web Services : présentation des procédures de sécurité (livre blanc)

Les bonnes pratiques de sécurité suivantes s'appliquent également à la configuration et à l'analyse des vulnérabilités dans Amazon DynamoDB :

- Surveillez la conformité de DynamoDB avec AWS Config Rules
- Surveillez la configuration DynamoDB avec AWS Config

Bonnes pratiques de sécurité pour Amazon DynamoDB

Amazon DynamoDB fournit différentes fonctions de sécurité à prendre en compte lorsque vous développez et implémentez vos propres politiques de sécurité. Les bonnes pratiques suivantes doivent être considérées comme des instructions générales et ne représentent pas une solution

de sécurité complète. Étant donné que ces bonnes pratiques peuvent ne pas être appropriées ou suffisantes pour votre environnement, considérez-les comme des remarques utiles plutôt que comme des recommandations.

Rubriques

- Bonnes pratiques de sécurité préventive pour DynamoDB
- Bonnes pratiques de sécurité de détection pour DynamoDB

Bonnes pratiques de sécurité préventive pour DynamoDB

Les bonnes pratiques suivantes peuvent vous aider à anticiper et à prévenir les incidents de sécurité dans Amazon DynamoDB.

Chiffrement au repos

DynamoDB chiffre au repos toutes les données utilisateur stockées dans des tables, index, flux et sauvegardes à l'aide de clés de chiffrement stockées dans AWS Key Management Service(AWS KMS). Cela fournit une couche supplémentaire de protection des données en sécurisant vos données contre tout accès non autorisé au stockage sous-jacent.

Vous pouvez spécifier si DynamoDB doit utiliser Clé détenue par AWS une (type de chiffrement par défaut), une clé gérée par le client ou Clé gérée par AWS une clé gérée par le client pour chiffrer les données utilisateur. Pour en savoir plus, consultez Chiffrement au repos Amazon DynamoDB.

Utiliser Des rôles IAM pour authentifier l'accès à DynamoDB

Pour que les utilisateurs, les applications et les autres AWS services puissent accéder à DynamoDB, ils doivent inclure des informations d'identification AWS valides dans AWS leurs demandes d'API. Vous ne devez pas stocker les AWS informations d'identification directement dans l'application ou l' EC2 instance. Il s'agit d'informations d'identification à long terme qui ne font pas l'objet d'une rotation automatique, et dont la compromission pourrait avoir un impact considérable sur l'activité. Un rôle IAM vous permet d'obtenir des clés d'accès temporaires qui peuvent être utilisées pour accéder aux AWS services et aux ressources.

Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour Amazon DynamoDB.

Utiliser les politiques IAM pour l'autorisation de base de DynamoDB

Lorsque vous accordez des autorisations, vous décidez qui les obtient, pour quel APIs DynamoDB ils obtiennent des autorisations et les actions spécifiques que vous souhaitez autoriser sur ces ressources. L'implémentation d'un privilège minimum est la clé de la réduction des risques de sécurité et de l'impact potentiel d'erreurs ou d'actes de malveillance.

Attachez des politiques d'autorisation à des identités IAM (c'est-à-dire des utilisateurs, des groupes et des rôles) et accordez ainsi des autorisations pour exécuter des opérations sur des ressources DynamoDB.

Pour ce faire, utilisez les ressources suivantes :

- AWS Politiques gérées (prédéfinies)
- Politiques gérées par le client

Utiliser des conditions de politique IAM pour un contrôle d'accès précis

Lorsque vous accordez des autorisations dans DynamoDB, vous pouvez spécifier des conditions pour déterminer comment une politique d'autorisation doit prendre effet. L'implémentation d'un privilège minimum est la clé de la réduction des risques de sécurité et de l'impact potentiel d'erreurs ou d'actes de malveillance.

Vous pouvez spécifier des conditions lors de l'octroi d'autorisations à l'aide d'une politique IAM. Par exemple, vous pouvez effectuer les opérations suivantes :

- Accordez des autorisations pour permettre aux utilisateurs d'accéder en lecture seule à certains éléments et attributs d'une table ou d'un index secondaire.
- Accordez des autorisations pour permettre aux utilisateurs d'accéder en écriture seule à certains attributs d'une table, en fonction de l'identité de cet utilisateur.

Pour plus d'informations, consultez Utilisation de conditions de politique IAM pour un contrôle d'accès précis.

Utiliser un point de terminaison et des politiques de VPC pour accéder à DynamoDB

Si vous n'avez besoin d'accéder à DynamoDB qu'à partir d'un cloud privé virtuel (VPC), vous devez utiliser un point de terminaison de VPC pour limiter l'accès uniquement à partir du VPC requis. Cela empêche ce trafic de passer par l'Internet ouvert et d'être sujet à cet environnement.

L'utilisation d'un point de terminaison de VPC pour DynamoDB vous permet de contrôler et de limiter l'accès à l'aide des ressources suivantes :

- Politiques de point de terminaison de VPC – Ces politiques sont appliquées sur le point de terminaison de VPC DynamoDB. Elles vous permettent de contrôler et de limiter l'accès d'API à la table DynamoDB.
- Politiques IAM – En utilisant la condition `aws:sourceVpce` sur des politiques attachées à des utilisateurs, groupes ou rôles, vous pouvez imposer que tout accès à la table DynamoDB se fasse via le point de terminaison de VPC spécifié.

Pour plus d'informations, consultez Points de terminaison pour Amazon DynamoDB.

Envisager un chiffrement côté client

Nous vous recommandons de planifier votre stratégie de chiffrement avant d'implémenter votre table dans DynamoDB. Si vous stockez des données sensibles ou confidentielles dans DynamoDB, pensez à inclure le chiffrement côté client dans votre stratégie. Vous pourrez ainsi chiffrer les données au plus près de leur origine et garantir leur protection tout au long de leur cycle de vie. Le chiffrement de vos données sensibles en transit et au repos contribue à garantir que vos données en texte brut ne sont pas accessibles à un tiers.

Le kit SDK de chiffrement de bases de données AWS pour DynamoDB est une bibliothèque logicielle qui vous aide à protéger vos données de table avant de les envoyer à DynamoDB. Il chiffre, signe, vérifie et déchiffre les éléments de votre table DynamoDB. Vous contrôlez les attributs qui sont chiffrés et signés.

Considérations relatives aux clés primaires

N'utilisez pas de noms sensibles ni de données sensibles en texte brut dans votre clé primaire pour votre table et vos index secondaires globaux. Les noms des clés apparaîtront dans la définition de votre table. Par exemple, les noms des clés primaires sont accessibles à toute personne autorisée à appeler DescribeTable. Les valeurs clés peuvent apparaître dans votre journal AWS CloudTrailet dans d'autres journaux. En outre, DynamoDB utilise les valeurs clés pour distribuer les données et acheminer les demandes AWS. Les administrateurs peuvent observer ces valeurs afin de préserver l'intégrité du service.

Si vous devez utiliser des données sensibles dans votre table ou des valeurs de clé GSI, nous vous recommandons d'utiliser end-to-end le chiffrement client. Cela vous permet d'effectuer des références clé-valeur à vos données tout en vous assurant qu'elles n'apparaissent jamais non chiffrées dans vos journaux liés à DynamoDB. Pour ce faire, vous pouvez utiliser le kit AWS SDK de chiffrement de base de données pour DynamoDB, mais cela n'est pas obligatoire. Si vous utilisez votre propre solution, nous devons toujours utiliser un algorithme de chiffrement

suffisamment sécurisé. Vous ne devez pas utiliser d'options non cryptographiques telles que le hachage, car elles ne sont pas considérées comme suffisamment sûres dans la plupart des cas.

Si les noms de vos clés primaires sont sensibles, nous vous recommandons d'utiliser plutôt `pk` et `sk`. Il s'agit d'une bonne pratique générale qui vous offre de la flexibilité dans la conception de votre clé de partition.

Consultez toujours vos experts en sécurité ou votre équipe chargée de votre AWS compte si vous vous demandez quel serait le bon choix.

Bonnes pratiques de sécurité de détection pour DynamoDB

Les bonnes pratiques suivantes pour Amazon DynamoDB peuvent vous aider à détecter des vulnérabilités de sécurité potentielles et autres incidents.

AWS CloudTrailÀ utiliser pour surveiller l'utilisation des clés KMS AWS gérées

Si vous utilisez un Clé gérée par AWSpour le chiffrement au repos, l'utilisation de cette clé est connectée AWS CloudTrail. CloudTrail fournit une visibilité sur l'activité des utilisateurs en enregistrant les actions effectuées sur votre compte. CloudTrail enregistre des informations importantes sur chaque action, notamment l'auteur de la demande, les services utilisés, les actions effectuées, les paramètres des actions et les éléments de réponse renvoyés par le AWS service. Ces informations vous aident à suivre les modifications apportées à vos AWS ressources et à résoudre les problèmes opérationnels. CloudTrail permet de garantir plus facilement le respect des politiques internes et des normes réglementaires.

Vous pouvez l'utiliser CloudTrail pour auditer l'utilisation des clés. CloudTrail crée des fichiers journaux contenant l'historique des appels d'AWS API et des événements associés à votre compte. Ces fichiers journaux incluent toutes les demandes d'AWS KMS API effectuées à l'AWS Management Console aide des outils de ligne de commande, en plus de celles effectuées via les AWS services intégrés.AWS SDKs Vous pouvez utiliser ces fichiers journaux pour obtenir des informations sur l'utilisation de la clé KMS, sur l'opération qui a été demandée, sur l'identité du demandeur, sur l'adresse IP à partir de laquelle la demande provenait, etc. Pour plus d'informations, consultez Journalisation des appels d'API AWS KMS avec AWS CloudTrail dans le Guide de l'utilisateur AWS CloudTrail.

Surveillez les opérations DynamoDB à l'aide de CloudTrail

CloudTrail peut surveiller à la fois les événements du plan de contrôle et les événements du plan de données. Les opérations de plan de contrôle vous permettent de créer et gérer des

tables DynamoDB. Elles vous permettent également d'utiliser les index, les flux et autres objets qui dépendent des tables. Les opérations de plan de données vous permettent d'exécuter des opérations de création, de lecture, de mise à jour et de suppression (également appelées actions CRUD) sur les données d'une table. Certaines opérations de plan de données vous permettent également de lire les données d'un index secondaire. Pour activer la journalisation des événements du plan de données dans CloudTrail, vous devez activer la journalisation de l'activité de l'API du plan de données dans CloudTrail. Pour plus d'informations, consultez Consignation d'événements de données pour les journaux d'activité.

Lorsqu'une activité se produit dans DynamoDB, cette activité est enregistrée dans CloudTrail un événement avec d'AWS autres événements de service dans l'historique des événements. Pour plus d'informations, consultez Journalisation des opérations DynamoDB à l'aide d'AWS CloudTrail. Vous pouvez consulter, rechercher et télécharger les événements récents dans votre AWS compte. Pour plus d'informations, consultez la section Affichage des événements avec l'historique des CloudTrail événements dans le guide de AWS CloudTrail l'utilisateur.

Pour un enregistrement continu des événements de votre AWS compte, y compris des événements pour DynamoDB, créez une trace. Un suivi permet CloudTrail de transférer des fichiers journaux vers un compartiment Amazon Simple Storage Service (Amazon S3). Par défaut, lorsque vous créez un parcours sur la console, celui-ci s'applique à toutes les AWS régions. Le journal d'activité consigne les événements de toutes les régions dans la partition AWS et livre les fichiers journaux dans le compartiment S3 de votre choix. En outre, vous pouvez configurer d'autres AWS services pour analyser plus en détail les données d'événements collectées dans les CloudTrail journaux et agir en conséquence.

Utiliser DynamoDB Streams pour surveiller des opérations de plan de données

DynamoDB est intégré afin que vous puissiez créer AWS Lambda des déclencheurs, des éléments de code qui répondent automatiquement aux événements dans DynamoDB Streams. Avec des déclencheurs, vous pouvez créer des applications qui réagissent à des modifications de données dans des tables DynamoDB.

Si vous activez DynamoDB Streams sur une table, vous pouvez associer l'ARN (Amazon Resource Name) de flux avec une fonction Lambda que vous écrivez. Immédiatement après la modification d'un élément de la table, un nouvel enregistrement apparaît dans le flux de la table.AWS Lambda interroge le flux et appelle votre fonction Lambda de manière synchrone lorsqu'elle détecte de nouveaux enregistrements de flux. La fonction Lambda peut effectuer toute action que vous spécifiez, comme envoyer une notification ou initier un flux de travail.

Pour voir un exemple, consultez Didacticiel : Utilisation d'AWS Lambda avec Amazon DynamoDB Streams. Cet exemple reçoit une entrée d'événement DynamoDB, traite les messages qu'elle contient et écrit certaines des données d'événement entrantes dans Amazon Logs. CloudWatch

Surveillez la configuration DynamoDB avec AWS Config

AWS Config vous permet de surveiller et d'enregistrer en permanence les changements de configuration de vos ressources AWS. Vous pouvez également l'utiliser AWS Config pour inventorier vos AWS ressources. Lors de la détection d'un changement par rapport à un état précédent, une notification Amazon Simple Notification Service (Amazon SNS) peut vous être envoyée pour vous permettre de vérifier et de réagir. Suivez les instructions de la section Configuration AWS Config avec la console, en vous assurant que les types de ressources DynamoDB sont inclus.

Vous pouvez configurer AWS Config pour diffuser les modifications de configuration et les notifications sur une rubrique Amazon SNS. Par exemple, lorsqu'une ressource est mise à jour, une notification peut être envoyée à votre adresse e-mail pour que vous puissiez consulter les modifications. Vous pouvez également être averti lors de l'AWS Configévaluation de vos règles personnalisées ou gérées par rapport à vos ressources.

Par exemple, consultez la rubrique Notifications AWS Config envoyées à un Amazon SNS dans le manuel du AWS Config développeur.

Surveiller la conformité de DynamoDB aux règles AWS Config

AWS Config suit en permanence les modifications de configuration qui se produisent parmi vos ressources. Il vérifie si ces changements ne vont pas à l'encontre de vos règles. Si une ressource enfreint une règle,AWS Config marque la ressource et la règle comme non conformes.

En utilisant AWS Config pour évaluer vos configurations de ressources, vous pouvez évaluer dans quelle mesure vos configurations de ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.AWS Config fournit des règles AWS gérées, qui sont des règles prédéfinies et personnalisables AWS Config utilisées pour évaluer si vos AWS ressources sont conformes aux meilleures pratiques courantes.

Etiqueter vos ressources DynamoDB pour l'identification et l'automatisation

Vous pouvez attribuer des métadonnées à vos AWS ressources sous forme de balises. Chaque étiquette est un libellé composé d'une clé définie par le client et d'une valeur facultative qui peut faciliter la gestion, la recherche et le filtrage de ressources.

L’étiquetage permet l’implémentation de contrôles groupés. Bien qu’il n’existe pas de types intrinsèques d’étiquettes, celles-ci vous permettent de catégoriser des ressources par objectif, par propriétaire, par environnement ou selon d’autres critères. Voici quelques exemples :

- Sécurité : utilisée pour déterminer des exigences telles que le chiffrement.
- Confidentialité – Identifiant pour le niveau spécifique de confidentialité des données qu’une ressource prend en charge.
- Environnement – Utilisé pour différencier les infrastructures de développement, de test et de production.

Pour plus d’informations, consultez Politiques d’étiquetage AWS et Étiquetage pour DynamoDB.

Surveillez votre utilisation d'Amazon DynamoDB en ce qui concerne les meilleures pratiques de sécurité en utilisant.AWS Security Hub CSPM

Security Hub CSPM utilise des contrôles de sécurité pour évaluer les configurations des ressources et les normes de sécurité afin de vous aider à vous conformer aux différents cadres de conformité.

Pour plus d'informations sur l'utilisation de Security Hub CSPM pour évaluer les ressources DynamoDB, consultez les contrôles Amazon DynamoDB dans le guide de l'utilisateur.AWS Security Hub CSPM

Surveillance et journalisation dans DynamoDB

La surveillance est un enjeu important pour assurer la fiabilité, la disponibilité et les performances de DynamoDB et de vos solutions AWS. Vous devez recueillir les données de surveillance de toutes les parties de vos solutions AWS de telle sorte que vous puissiez déboguer plus facilement une éventuelle défaillance à plusieurs points.

Rubriques

- Plan de surveillance
- Référence des performances
- Services intégrés
- Outils de surveillance automatique
- Surveillance des métriques dabs DynamoDB avec Amazon CloudWatch
- Journalisation des opérations de DynamoDB à l’aide d’AWS CloudTrail
- Analyse de l'accès aux données à l'aide des informations des CloudWatch contributeurs pour DynamoDB

Plan de surveillance

Avant de commencer à surveiller DynamoDB, créez un plan de surveillance contenant les réponses aux questions suivantes :

- Quels sont les objectifs de la surveillance ?
- Quelles sont les ressources à surveiller ?
- À quelle fréquence les ressources doivent-elles être surveillées ?
- Quels outils de surveillance utiliser ?
- Qui exécute les tâches de supervision ?
- Qui doit être informé en cas de problème ?

Référence des performances

Établissez une référence de performances normales de DynamoDB dans votre environnement, en mesurant les performances à divers moments et dans diverses conditions de charge. Lorsque vous

surveillez DynamoDB, songez à stocker l'historique des données de surveillance. Ces données stockées constituent une référence pour comparer des données de performances actuelles, identifier des modèles de performance normaux et des anomalies de performance, et concevoir des méthodes pour résoudre des problèmes. Pour établir une référence, vous devez, au moins, superviser les éléments suivants :

- Nombre d'unités de capacité de lecture ou d'écriture consommées sur la période spécifiée, de sorte que vous puissiez suivre la quantité de votre débit provisionné utilisée.
- Demandes qui ont dépassé la capacité d'écriture ou de lecture allouée d'une table pendant la période spécifiée, afin de déterminer les demandes qui dépassent les quotas de débit alloué d'une table.
- Les erreurs système, afin de pouvoir déterminer si des demandes ont entraîné une erreur.

Services intégrés

DynamoDB surveille automatiquement les tables pour vous et présente les métriques via Amazon CloudWatch. En outre, DynamoDB s'intègre avec les Services AWS suivants pour vous aider à surveiller, déboguer et dépanner vos ressources DynamoDB.

- AWS CloudTrail capture les appels d'API et les événements associés créés par votre Compte AWS ou au nom de celui-ci et livre les fichiers journaux dans un compartiment Amazon S3 que vous spécifiez. Pour plus d'informations, consultez Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail.
- Contributor Insights est un outil de diagnostic pour identifier rapidement les clés plus consultées et les plus limitées dans votre table. Pour plus d'informations, consultez Analyse de l'accès aux données à l'aide des informations des CloudWatch contributeurs pour DynamoDB.

Outils de surveillance automatique

AWS fournit divers outils que vous pouvez utiliser pour surveiller DynamoDB. Nous vous recommandons d'automatiser le plus possible les tâches de supervision. Pour surveiller DynamoDB et signaler des problèmes, vous pouvez utiliser les outils de surveillance automatisée suivants :

- Alarmes Amazon CloudWatch : surveillez une seule métrique sur une période définie et exécutez une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné sur un certain nombre de périodes.

L'action est une notification envoyée à une rubrique Amazon Simple Notification Service (Amazon SNS) ou une politique Amazon EC2 Auto Scaling. Les alarmes Amazon CloudWatch n'appellent pas d'actions simplement parce qu'elles sont dans un état particulier : l'état doit avoir changé et été maintenu pendant un certain nombre de périodes. Pour plus d'informations, consultez Surveillance des métriques dabs DynamoDB avec Amazon CloudWatch.

- Surveillance des journaux AWS CloudTrail : partagez des fichiers journaux entre comptes, surveillez les fichiers journaux AWS CloudTrail en temps réel en les envoyant dans des journauxAWS CloudTrail, écrivez des applications de traitement des journaux en Java et vérifiez que vos fichiers journaux n'ont pas changé après leur livraison par AWS CloudTrail. Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch Logs ? dans le Guide de l'utilisateur AWS CloudTrail.

Surveillance des métriques dabs DynamoDB avec Amazon CloudWatch

Vous pouvez surveiller DynamoDB à l'aide de CloudWatch, qui collecte et convertit les données brutes de DynamoDB en métriques lisibles et disponibles en quasi-temps réel. Ces statistiques étant conservées pendant un certain temps, vous pouvez accéder à des informations historiques pour acquérir une meilleure perspective sur les performances de votre service ou application web. Par défaut, les données des métriques de DynamoDB sont automatiquement envoyées à CloudWatch. Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch ? dans Conservation des métriques dans le Guide de l'utilisateur Amazon CloudWatch.

Rubriques

- Comment utiliser les métriques de DynamoDB ?
- Affichage des métriques dans la console CloudWatch
- Affichage des métriques dans la AWS CLI
- Métriques et dimensions DynamoDB
- Création d' CloudWatch alarmes dans DynamoDB

Comment utiliser les métriques de DynamoDB ?

Les métriques rapportées par DynamoDB fournissent des informations que vous pouvez analyser de différentes manières. La liste suivante présente certaines utilisations courantes des métriques. Voici quelques suggestions pour vous aider à démarrer, qui ne forment pas une liste exhaustive.

Comment utiliser les métriques de DynamoDB ?

Comment puis-je ?	Métriques pertinentes
Comment surveiller le taux de suppressions par TTL sur ma table ?	Vous pouvez surveiller `TimeToLiveDeletedItemCount` sur la période spécifiée afin de suivre le taux des suppressions par TTL sur votre table. Pour un exemple d'application sans serveur utilisant les métriques de `TimeToLiveDeletedItemCount`, consultez Automatically archive items to S3 using DynamoDB time to live (TTL) with AWS Lambda and Amazon Data Firehose.
Comment puis-je déterminer la part de mon débit provisionné qui est utilisée ?	Vous pouvez surveiller les valeurs `ConsumedReadCapacityUnits` ou `ConsumedWriteCapacityUnits` sur la période spécifiée afin de suivre la quantité de votre débit provisionné utilisée.
Comment puis-je déterminer les demandes qui dépassent les quotas de débit provisionné d'une table ?	`ThrottledRequests` est incrémenté de 1 si les événements au sein d'une demande dépassent un quota de débit provisionné. Ensuite, pour obtenir une information sur l'événement qui limite une demande, comparez la valeur `ThrottledRequests` avec les métriques `ReadThrottleEvents` et `WriteThrottleEvents` pour la table et ses index.
Comment déterminer si des erreurs système se sont produites	Vous pouvez surveiller `SystemErrors` pour déterminer si des demandes ont entraîné un code HTTP 500 (erreur de serveur). En règle générale, cette métrique doit être égale à zéro. Si tel n'est pas le cas, vous pouvez enquêter.
Comment surveiller la valeur de latences pour les opérations de mes tables ?	Vous pouvez surveiller `SuccessfulRequestLatency` en suivant la latence moyenne et la latence médiane à l'aide de mesures percentiles (p50). Les pics de latence occasionnels ne sont pas préoccupants. Toutefois, si la latence moyenne ou p50

Comment puis-je ?	Métriques pertinentes
	(médiane) est élevée, il se peut qu'il y ait un problème sous-jacent que vous devez résoudre. Pour plus d'informations, consultez Résolution des problèmes de latence dans Amazon DynamoDB.

Affichage des métriques dans la console CloudWatch

Les métriques sont d'abord regroupées par espace de noms de service, puis par les différentes combinaisons de dimension au sein de chaque espace de noms.

Pour afficher les métriques sur la console CloudWatch

1. Ouvrez la console CloudWatch à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Dans le panneau de navigation, sélectionnez Métriques, Toutes les métriques.
3. Sélectionnez l'espace de noms DynamoDB. Vous pouvez également sélectionner l'espace de noms Utilisation pour afficher les métriques d'utilisation de DynamoDB. Pour plus d'informations sur les métriques, consultez Métriques d'utilisation d'AWS.
4. L'onglet Parcourir affiche toutes les métriques dans l'espace de noms.
5. (Facultatif) Pour ajouter graphique de graphique à un tableau de bord CloudWatch, choisissez Actions, Ajouter au tableau de bord.

Affichage des métriques dans la AWS CLI

Pour plus d'informations sur les métriques utilisant la AWS CLI, utilisez la commande CloudWatch `list-metrics`. Dans l'exemple indiqué ci-dessous, vous répertoriez toutes les métriques dans l'espace de noms `AWS/DynamoDB`.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Pour obtenir des statistiques de métriques, utilisez la commande`get-metric-statistics`. La commande suivante obtient des statistiques `ConsumedReadCapacityUnits` pour la table `ProductCatalog` au cours d'une période de 24 heures donnée, avec une précision d'un niveau de 5 minutes.

```
aws cloudwatch get-metric-statistics –namespace AWS/DynamoDB \
     –metric-name ConsumedReadCapacityUnits \
     –start-time 2023-11-01T00:00:00Z \
     –end-time 2023-11-02T00:00:00Z \
     –period 360 \
     –statistics Average \
     –dimensions Name=TableName,Value=ProductCatalog
```

L’exemple de sortie apparaît comme suit :

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

Métriques et dimensions DynamoDB

Lorsque vous interagissez avec DynamoDB, celui-ci envoie des métriques et des dimensions à. CloudWatch

DynamoDB produit le débit provisionné consommé pendant des périodes d'une minute. La mise à l'échelle automatique se déclenche lorsque la capacité consommée dépasse l'utilisation cible configurée pendant deux minutes consécutives. CloudWatch les alarmes peuvent avoir un court délai de quelques minutes avant de déclencher le redimensionnement automatique. Ce délai garantit une évaluation CloudWatch précise des mesures. Si les pics de débit consommés sont espacés de plus d'une minute, l'autoscaling risque de ne pas se déclencher. De même, un événement de réduction verticale peut être déclenché lorsque 15 points de données consécutifs sont en dessous de l'utilisation cible. Dans les deux cas, après le déclenchement de l'autoscaling, l'UpdateTableAPI est invoquée. Plusieurs minutes sont ensuite nécessaires pour mettre à jour la capacité provisionnée pour la table ou l'index. Pendant cette période, toutes les demandes dépassant la capacité provisionnée précédente pour les tables seront limitées.

Affichage des métriques et dimensions

CloudWatch affiche les mesures suivantes pour DynamoDB :

Métriques DynamoDB

> **Note**
>
> Amazon CloudWatch agrège ces statistiques à intervalles d'une minute :
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadAccountLimitThrottleEvents`
> - `ReadKeyRangeThroughputThrottleEvents`
> - `ReadMaxOnDemandThroughputThrottleEvents`
> - `ReadProvisionedThroughputThrottleEvents`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`

- `ThrottledRequests`
- `TransactionConflict`
- `UserErrors`
- `WriteAccountLimitThrottleEvents`
- `WriteKeyRangeThroughputThrottleEvents`
- `WriteMaxOnDemandThroughputThrottleEvents`
- `WriteProvisionedThroughputThrottleEvents`
- `WriteThrottleEvents`

Pour toutes les autres métriques de DynamoDB, la granularité d'agrégation est de cinq minutes.

Certaines statistiques, telles que Moyenne ou Somme, s'appliquent à chaque métrique. Cependant, toutes ces valeurs sont disponibles via la console Amazon DynamoDB, ou à l'aide de la console, ou pour toutes CloudWatch les métriques AWS SDKs .AWS CLI

Dans la liste suivante, chaque métrique dispose d'un ensemble de statistiques valides qui s'appliquent à cette métrique.

Répertorier les métriques disponibles

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount

- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization
- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadAccountLimitThrottleEvents
- ReadKeyRangeThroughputThrottleEvents
- ReadMaxOnDemandThroughputThrottleEvents
- ReadProvisionedThroughputThrottleEvents
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents
- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents

- WriteThrottleEvents
- Métriques d'utilisation

AccountMaxReads

Nombre maximum d'unités de capacité de lecture qu'un compte peut utiliser. Cette limite ne s'applique pas aux tables à la demande ou aux index secondaires globaux.

Unités `Count`:

Statistiques valides :

- `Maximum` – Nombre maximum d'unités de capacité de lecture qu'un compte peut utiliser.

AccountMaxTableLevelReads

Nombre maximum d'unités de capacité de lecture qu'une table ou un index secondaire global d'un compte peuvent utiliser. Pour des tables à la demande, cette limite plafonne le nombre maximum d'unités de demande de lecture qu'une table ou un index secondaire global peuvent utiliser.

Unités `Count`:

Statistiques valides :

- `Maximum` – Nombre maximum d'unités de capacité de lecture qu'une table ou un index secondaire global du compte peuvent utiliser.

AccountMaxTableLevelWrites

Nombre maximum d'unités de capacité d'écriture qu'une table ou un index secondaire global d'un compte peuvent utiliser. Pour des tables à la demande, cette limite plafonne le nombre maximum d'unités de demande d'écriture qu'une table ou un index secondaire global peuvent utiliser.

Unités `Count`:

Statistiques valides :

- `Maximum` – Nombre maximum d'unités de capacité d'écriture qu'une table ou un index secondaire global du compte peuvent utiliser.

AccountMaxWrites

Nombre maximum d’unités de capacité d’écriture qu’un compte peut utiliser. Cette limite ne s’applique pas aux tables à la demande ou aux index secondaires globaux.

Unités `Count`:

Statistiques valides :

- `Maximum` – Nombre maximum d'unités de capacité d'écriture qu'un compte peut utiliser.

AccountProvisionedReadCapacityUtilization

Pourcentage d'unités de capacité de lecture approvisionnée qu'un compte utilise.

Unités : `Percent`

Statistiques valides :

- `Maximum` – Pourcentage maximum d’unités de capacité de lecture approvisionnée que le compte utilise.
- `Minimum` – Pourcentage minimum d’unités de capacité de lecture approvisionnée que le compte utilise.
- `Average` – Pourcentage moyen d’unités de capacité de lecture approvisionnée que le compte utilise. La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité de lecture approvisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

AccountProvisionedWriteCapacityUtilization

Pourcentage d'unités de capacité d'écriture approvisionnée qu'un compte utilise.

Unités : `Percent`

Statistiques valides :

- `Maximum` – Pourcentage maximum d’unités de capacité d’écriture approvisionnée que le compte utilise.
- `Minimum` – Pourcentage minimum d’unités de capacité d’écriture approvisionnée que le compte utilise.

- `Average` – Pourcentage moyen d'unités de capacité d'écriture approvisionnée que le compte utilise. La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité d'écriture approvisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

AgeOfOldestUnreplicatedRecord

Temps écoulé depuis qu'un registre devant encore être répliqué vers le flux de données Kinesis est apparu pour la première fois dans la table DynamoDB.

Unités : `Milliseconds`

Dimensions : `TableName, DelegatedOperation`

Statistiques valides :

- `Maximum`.
- `Minimum`.
- `Average`.

ConditionalCheckFailedRequests

Nombre de tentatives infructueuses d'exécution d'écritures conditionnelles. Les opérations `PutItem`, `UpdateItem` et `DeleteItem` vous permettent de fournir une condition logique dont l'évaluation doit être true avant que l'opération puisse continuer. Si l'évaluation de cette condition est false, `ConditionalCheckFailedRequests` est incrémenté d'une unité. `ConditionalCheckFailedRequests` est également incrémenté d'une unité pour les instructions PartiQL Update et Delete qui incluent une condition logique dont l'évaluation est false.

> **Note**
>
> Un échec d'écriture conditionnelle entraîne une erreur HTTP 400 (Demande incorrecte). Ces événements sont reflétés dans la métrique `ConditionalCheckFailedRequests`, mais pas dans la métrique `UserErrors`.

Unités : `Count`

Dimensions : `TableName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

Nombre d'unités de récupération de données de modification consommées.

Unités : `Count`

Dimensions : `TableName, DelegatedOperation`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

Nombre d’unités de capacité de lecture consommées sur la période spécifiée pour la capacité à la demande et provisionnée, de sorte que vous puissiez suivre la quantité de votre débit utilisée. Vous pouvez extraire la capacité totale de lecture consommée pour une table et tous ses index secondaires globaux, ou pour un index secondaire global particulier. Pour de plus amples informations, veuillez consulter Mode de capacité en lecture/écriture.

La dimension `TableName` renvoie la valeur `ConsumedReadCapacityUnits` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `ConsumedReadCapacityUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

> **Note**
>
> Cela signifie que les pics courts et intenses de consommation de capacité d'une seconde seulement peuvent ne pas être reflétés avec précision dans le CloudWatch graphique, ce qui peut entraîner une baisse du taux de consommation apparent pendant cette minute. Utilisez la statistique `Sum` pour calculer le débit consommé. Par exemple, obtenez la valeur `Sum` sur une durée d'une minute, puis divisez-la par le nombre de secondes dans une minute (60) afin de calculer la valeur moyenne de `ConsumedReadCapacityUnits` par seconde. Vous pouvez comparer la valeur calculée à la valeur de débit approvisionné que vous fournissez à DynamoDB.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Nombre minimum d'unités de capacité de lecture consommées par une demande adressée à la table ou à l'index.
- `Maximum` – Nombre maximum d'unités de capacité de lecture consommées par une demande adressée à la table ou à l'index.
- `Average` – Capacité de lecture moyenne par demande consommée.

  > **Note**
  >
  > La valeur `Average` est influencée par des périodes d'inactivité où la valeur de l'échantillon est nulle.

- `Sum` – Nombre total d'unités de capacité de lecture consommées. Il s'agit de la statistique la plus utile pour la métrique `ConsumedReadCapacityUnits`.
- `SampleCount` – Représente la fréquence à laquelle la métrique est émise. Même pour les tables dont le trafic est nul, la valeur `SampleCount` est régulièrement émise, mais les valeurs d'échantillon seront toujours nulles.

> **Note**
>
> La valeur `SampleCount` est influencée par des périodes d'inactivité où la valeur de l'échantillon est nulle.

ConsumedWriteCapacityUnits

Nombre d'unités de capacité d'écriture consommées sur la période spécifiée pour la capacité à la demande et provisionnée, de sorte que vous puissiez suivre la quantité de votre débit utilisée. Vous pouvez extraire la capacité totale d'écriture consommée pour une table et tous ses index secondaires globaux, ou pour un index secondaire global particulier. Pour de plus amples informations, veuillez consulter Mode de capacité en lecture/écriture.

La dimension `TableName` renvoie la valeur `ConsumedWriteCapacityUnits` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `ConsumedWriteCapacityUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

> **Note**
>
> Utilisez la statistique `Sum` pour calculer le débit consommé. Par exemple, récupérez la valeur `Sum` sur une durée d'une minute, puis divisez-la par le nombre de secondes dans une minute (60) afin calculer la valeur `ConsumedWriteCapacityUnits` moyenne par seconde (en sachant que cette moyenne ne met pas en évidence les grands pics mais les pic brefs d'activité d'écriture pendant cette minute). Vous pouvez comparer la valeur calculée à la valeur de débit approvisionné que vous fournissez à DynamoDB.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Nombre minimum d'unités de capacité d'écriture consommées par une demande adressée à la table ou à l'index.

- `Maximum` – Nombre maximum d'unités de capacité d'écriture consommées par une demande adressée à la table ou à l'index.
- `Average` – Capacité d'écriture moyenne par demande consommée.

> **Note**
>
> La valeur `Average` est influencée par des périodes d'inactivité où la valeur de l'échantillon est nulle.

- `Sum` – Nombre total d'unités de capacité d'écriture consommées. Il s'agit de la statistique la plus utile pour la métrique `ConsumedWriteCapacityUnits`.
- `SampleCount` – Représente la fréquence à laquelle la métrique est émise. Même pour les tables dont le trafic est nul, la valeur `SampleCount` est régulièrement émise, mais les valeurs d'échantillon seront toujours nulles.

> **Note**
>
> La valeur `SampleCount` est influencée par des périodes d'inactivité où la valeur de l'échantillon est nulle.

FailedToReplicateRecordCount

Nombre de registres que DynamoDB n'a pas pu répliquer sur votre flux de données Kinesis.

Unités : `Count`

Dimensions: `TableName`, `DelegatedOperation`

Statistiques valides :

- `Sum`

MaxProvisionedTableReadCapacityUtilization

Pourcentage de capacité de lecture allouée qu'utilisent la table de lecture ou l'index secondaire global les plus alloués d'un compte.

Unités `Percent`:

Statistiques valides :

- `Maximum` – Le pourcentage maximum d'unités de capacité de lecture allouée qu'utilisent la table de lecture ou l'index secondaire global les plus alloués d'un compte.
- `Minimum` – Le pourcentage minimum d'unités de capacité de lecture allouée qu'utilisent la table de lecture ou l'index secondaire global les plus alloués d'un compte.
- `Average` – Le pourcentage moyen d'unités de capacité de lecture allouée qu'utilisent la table de lecture ou l'index secondaire global les plus alloués du compte. La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité de lecture approvisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

MaxProvisionedTableWriteCapacityUtilization

Pourcentage de capacité d'écriture approvisionnée qu'utilisent la table d'écriture ou l'index secondaire global les plus approvisionnés d'un compte.

Unités : `Percent`

Statistiques valides :

- `Maximum` – Pourcentage maximum d'unités de capacité d'écriture approvisionnée qu'utilise la table d'écriture ou l'index secondaire global les plus approvisionnés d'un compte.
- `Minimum` – Pourcentage minimum d'unités de capacité d'écriture approvisionnée qu'utilise la table d'écriture ou l'index secondaire global les plus approvisionnés d'un compte.
- `Average` – Pourcentage moyen d'unités de capacité d'écriture approvisionnée qu'utilise la table d'écriture ou l'index secondaire global les plus approvisionnés du compte. La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité d'écriture approvisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

OnDemandMaxReadRequestUnits

Nombre d'unités de demande de lecture à la demande pour une table ou un index secondaire global.

Pour afficher la valeur `OnDemandMaxReadRequestUnits` pour une table, vous devez spécifier `TableName`. Pour afficher la valeur `OnDemandMaxReadRequestUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : nombre

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Paramètre le plus bas pour les unités de demande de lecture à la demande. Si vous utilisez `UpdateTable` pour augmenter les unités de demande de lecture, cette métrique indique la valeur `ReadRequestUnits` à la demande la plus faible pendant cette période.
- `Maximum` – Paramètre le plus élevé pour les unités de demande de lecture à la demande. Si vous utilisez `UpdateTable` pour diminuer les unités de demande de lecture, cette métrique indique la valeur `ReadRequestUnits` à la demande la plus élevée pendant cette période.
- `Average` – Nombre moyen d'unités de demande de lecture à la demande. La métrique `OnDemandMaxReadRequestUnits` est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de demande de lecture à la demande, il se peut que cette statistique ne reflète pas la moyenne réelle.

OnDemandMaxWriteRequestUnits

Nombre d'unités de demande d'écriture à la demande pour une table ou un index secondaire global.

Pour afficher la valeur `OnDemandMaxWriteRequestUnits` pour une table, vous devez spécifier `TableName`. Pour afficher la valeur `OnDemandMaxWriteRequestUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Paramètre le plus bas pour les unités de demande d'écriture à la demande. Si vous utilisez `UpdateTable` pour augmenter les unités de demande d'écriture, cette métrique indique la valeur `WriteRequestUnits` à la demande la plus faible pendant cette période.
- `Maximum` – Paramètre le plus élevé pour les unités de demande d'écriture à la demande. Si vous utilisez `UpdateTable` pour diminuer les unités de demande d'écriture, cette métrique indique la valeur `WriteRequestUnits` à la demande la plus élevée pendant cette période.
- `Average` – Nombre moyen d'unités de demande d'écriture à la demande. La métrique `OnDemandMaxWriteRequestUnits` est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de demande d'écriture à la demande, il se peut que cette statistique ne reflète pas la moyenne réelle.

OnlineIndexConsumedWriteCapacity

Nombre d'unités de capacité d'écriture consommées lors de l'ajout d'un nouvel index secondaire global à une table. Si la capacité d'écriture de l'index est trop faible, l'activité d'écriture entrante pendant la phase de remplissage peut être limitée. Cela peut augmenter le temps nécessaire pour créer l'index. Vous devez surveiller cette statistique pendant la création de l'index afin de déterminer si la capacité d'écriture de l'index est insuffisamment approvisionnée.

Vous pouvez ajuster la capacité d'écriture de l'index à l'aide de l'opération `UpdateTable`, même si l'index est encore en cours de génération.

La métrique `ConsumedWriteCapacityUnits` pour l'index n'inclut pas le débit d'écriture consommé pendant de la création de l'index.

> **Note**
>
> Cette métrique peut ne pas être émise si la phase de remplissage du nouvel index secondaire global se termine rapidement (en moins de quelques minutes). Cela peut se produire si la table de base ne contient que peu ou pas d'éléments à remplir dans l'index.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

Pourcentage d'accomplissement lors de l'ajout un nouvel index secondaire global à une table. DynamoDB doit d'abord allouer des ressources pour le nouvel index, puis remplir les attributs de la table dans l'index. Pour les tables de grande taille, ce processus peut prendre beaucoup de temps.

Vous devez surveiller cette statistique pour suivre la progression de la génération de l’index par DynamoDB.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

Nombre d'événements de limitation d'écriture qui se produisent lors de l'ajout d'un nouvel index secondaire global à une table. Ces événements indiquent que la création d’index prendra plus de temps, car l’activité d’écriture entrante dépasse le débit d’écriture approvisionné de l’index.

Vous pouvez ajuster la capacité d’écriture de l’index à l’aide de l’opération `UpdateTable`, même si l’index est encore en cours de génération.

La métrique `WriteThrottleEvents` pour l’index n’inclut pas les événements de limitation qui se produisent lors de la création de l’index.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Métrique pour Tables globales de la version 2017.11.29 (héritée) (tables globales uniquement). Nombre de mises à jour d'éléments écrites dans une table de réplique, mais pas encore écrites dans un autre réplica dans la table globale.

Unités : `Count`

Dimensions : `TableName, ReceivingRegion`

Statistiques valides :

- `Average`
- `Sample Count`
- `Sum`

ProvisionedReadCapacityUnits

Nombre d'unités de capacité de lecture approvisionnée pour une table ou un index secondaire global. La dimension `TableName` renvoie la valeur `ProvisionedReadCapacityUnits` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `ProvisionedReadCapacityUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Paramètre le plus bas pour la capacité de lecture approvisionnée. Si vous utilisez `UpdateTable` pour augmenter la capacité de lecture, cette métrique indique la valeur `ReadCapacityUnits` approvisionnée la plus faible pendant cette période.
- `Maximum` – Paramètre le plus élevé pour la capacité de lecture approvisionnée. Si vous utilisez `UpdateTable` pour réduire la capacité de lecture, cette métrique indique la valeur `ReadCapacityUnits` approvisionnée la plus élevée pendant cette période.
- `Average` – Capacité de lecture approvisionnée moyenne. La métrique `ProvisionedReadCapacityUnits` est publiée à intervalles de cinq minutes. Par conséquent,

si vous ajustez rapidement les unités de capacité de lecture approvisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

ProvisionedWriteCapacityUnits

Nombre d'unités de capacité d'écriture approvisionnée pour une table ou un index secondaire global.

La dimension `TableName` renvoie la valeur `ProvisionedWriteCapacityUnits` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `ProvisionedWriteCapacityUnits` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Minimum` – Paramètre le plus bas pour la capacité d'écriture approvisionnée. Si vous utilisez `UpdateTable` pour augmenter la capacité d'écriture, cette métrique indique la valeur `WriteCapacityUnits` approvisionnée la plus faible pendant cette période.
- `Maximum` – Paramètre le plus élevé pour la capacité d'écriture approvisionnée. Si vous utilisez `UpdateTable` pour réduire la capacité d'écriture, cette métrique indique la valeur `WriteCapacityUnits` approvisionnée la plus élevée pendant cette période.
- `Average` – Capacité d'écriture approvisionnée moyenne. La métrique `ProvisionedWriteCapacityUnits` est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité d'écriture provisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

ReadAccountLimitThrottleEvents

Nombre de demandes de lecture limitées en raison des limites du compte.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.

- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

ReadKeyRangeThroughputThrottleEvents

Nombre de demandes de lecture limitées en raison des limites de partition.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

ReadMaxOnDemandThroughputThrottleEvents

Nombre de demandes de lecture limitées en raison du débit maximal à la demande.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

ReadProvisionedThroughputThrottleEvents

Nombre de demandes de lecture limitées en raison de limites de débit approvisionné.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

ReadThrottleEvents

Demandes adressées à DynamoDB qui dépassent le nombre d'unités de capacité de lecture approvisionnée pour une table ou un index secondaire global.

Une seule demande peut entraîner plusieurs événements. Par exemple, une opération `BatchGetItem` qui lit 10 éléments est traitée comme 10 événements `GetItem`. Pour chaque événement, `ReadThrottleEvents` est incrémenté d'une unité si l'événement est limité. La métrique `ThrottledRequests` pour l'opération `BatchGetItem` entière n'est incrémentée que si les 10 événements `GetItem` sont limités.

La dimension `TableName` renvoie la valeur `ReadThrottleEvents` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `ReadThrottleEvents` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `SampleCount`
- `Sum`

ReplicationLatency

(Cette métrique a trait aux tables globales DynamoDB.) Temps écoulé entre la mise à jour d'un élément apparaissant dans le flux DynamoDB pour une table de réplique, et l'affichage de cet élément dans un autre réplica dans la table globale.

Unités : `Milliseconds`

Dimensions : `TableName, ReceivingRegion`

Statistiques valides :

- `Average`
- `Minimum`
- `Maximum`

ReturnedBytes

Nombre d'octets renvoyés par des opérations `GetRecords` (Amazon DynamoDB Streams) durant la période spécifiée.

Unités : `Bytes`

Dimensions : `Operation, StreamLabel, TableName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

Nombre de lignes renvoyées par (certaines) opérations `Query`, `Scan` ou `ExecuteStatement` durant la période spécifiée.

Le nombre d’éléments renvoyés n’est pas nécessairement le même que le nombre d’éléments évalués. Par exemple, supposons que vous ayez demandé une opération `Scan` sur une table ou un index contenant 100 éléments, mais spécifié une valeur `FilterExpression` réduisant les résultats de sorte que seuls 15 éléments ont été retournés. Dans ce cas, la réponse de l’opération `Scan` contient une valeur `ScanCount` de 100 et une valeur `Count` de 15 éléments renvoyés.

Unités : `Count`

Dimensions : `TableName, Operation`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

Nombre de registres de flux renvoyés par des opérations `GetRecords` (Amazon DynamoDB Streams) durant la période spécifiée.

Unités : `Count`

Dimensions : `Operation, StreamLabel, TableName`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

Latence des demandes réussies adressées à DynamoDB ou à Amazon DynamoDB Streams durant la période spécifiée. `SuccessfulRequestLatency` peut fournir deux types différents d'informations :

- Temps écoulé pour les demandes réussies (`Minimum`, `Maximum`, `Sum`, `Average` ou `Percentile`).

- Nombre de demandes réussies (`SampleCount`).

`SuccessfulRequestLatency` reflète l'activité uniquement au sein de DynamoDB ou d'Amazon DynamoDB Streams, et ne prend en compte ni la latence réseau, ni l'activité côté client.

> **Note**
>
> Pour analyser des valeurs de percentile personnalisées (telles que p99,9), vous pouvez saisir manuellement le percentile souhaité (par exemple, p99,9) dans le champ de statistiques métriques. CloudWatch Cela vous permet d'évaluer les distributions de latence au-delà des percentiles par défaut répertoriés dans la liste déroulante.

Unités : `Milliseconds`

Dimensions : `TableName, Operation, StreamLabel`

Statistiques valides :

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

Demandes adressées à DynamoDB ou à Amazon DynamoDB Streams qui génèrent un code d'état HTTP 500 durant la période spécifiée. Un état HTTP 500 indique généralement une erreur de service interne.

> **Note**
>
> Lorsque DynamoDB renvoie une erreur système (HTTP 500), la AWS SDKs plupart exécutent automatiquement un nombre de tentatives configurable. Si le problème est résolu

lors d'une nouvelle tentative, votre application continue sans que l'erreur ne s'affiche, et vous remarquerez peut-être une augmentation de la latence perçue côté client. Si l'erreur persiste après toutes les tentatives, elle se propage dans le code de votre application.

Unités : `Count`

Dimensions : `TableName, Operation`

Statistiques valides :

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

Nombre d'éléments supprimés par TTL pendant la période spécifiée. Cette métrique vous aide à surveiller le taux de suppressions par TTL sur votre table.

Unités : `Count`

Dimensions : TableName

Statistiques valides :

- `Sum`

ThrottledPutRecordCount

Le nombre de registres qui ont été limités sur votre flux de données Kinesis en raison d'une capacité insuffisante de Kinesis Data Streams.

Unités : `Count`

Dimensions : TableName, DelegatedOperation

Statistiques valides :

- `Minimum`

- `Maximum`
- `Average`
- `SampleCount`

ThrottledRequests

Demandes adressées à DynamoDB qui dépassent les limites de débit approvisionnées sur une ressource (telle qu'une table ou un index).

`ThrottledRequests` est incrémenté d'une unité si un événement au sein d'une demande dépasse une limite de débit approvisionné. Par exemple, si vous mettez à jour un élément dans une table ayant des index secondaires globaux, cela correspond à plusieurs événements : une écriture dans la table et une écriture dans chaque index. Si un ou plusieurs de ces événements sont limités, `ThrottledRequests` est incrémenté d'une unité.

> **Note**
>
> Dans une demande de lot (`BatchGetItem` ou `BatchWriteItem`), `ThrottledRequests` n'est incrémenté que si chaque demande dans le lot est limitée.
> Si une demande dans le lot est limitée, l'une des métriques suivantes est incrémentée :
>
> - `ReadThrottleEvents` – Pour un événement `GetItem` limité dans `BatchGetItem`.
> - `WriteThrottleEvents` – Pour un événement `PutItem` ou `DeleteItem` limité dans `BatchWriteItem`.

Pour obtenir une information sur l'événement qui limite une demande, comparez la valeur `ThrottledRequests` avec les valeurs `ReadThrottleEvents` et `WriteThrottleEvents` pour la table et ses index.

> **Note**
>
> Une demande limitée génère un code d'état HTTP 400. Tous ces événements sont reflétés dans la métrique `ThrottledRequests`, mais pas dans la métrique `UserErrors`.

Unités : `Count`

Dimensions : `TableName, Operation`

Statistiques valides :

- `Sum`
- `SampleCount`

TransactionConflict

Demandes au niveau de l'élément rejetées en raison de conflits transactionnels entre des demandes simultanées sur les mêmes éléments. Pour plus d'informations, consultez Gestion des conflits de transaction dans DynamoDB.

Unités : `Count`

Dimensions : `TableName`

Statistiques valides :

- `Sum` – Nombre de demandes au niveau de l'élément rejetées en raison de conflits de transactions.

  > **Note**
  >
  > Si plusieurs demandes au niveau de l'élément dans un appel à `TransactWriteItems` ou à `TransactGetItems` ont été rejetées, la valeur `Sum` est incrémentée d'une unité pour chaque demande `Put`, `Update`, `Delete` ou `Get` au niveau de l'élément.

- `SampleCount` – Nombre de demandes rejetées en raison de conflits de transactions.

  > **Note**
  >
  > Si plusieurs demandes au niveau de l'élément dans un appel à `TransactWriteItems` ou à `TransactGetItems` ont été rejetées, la valeur `SampleCount` n'est incrémentée que d'une unité.

- `Min` – Nombre minimum de demandes au niveau de l'élément rejetées dans un appel à `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.
- `Max` – Nombre maximum de demandes au niveau de l'élément rejetées dans un appel à `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.

- `Average` – Nombre moyen de demandes au niveau de l'élément rejetées dans un appel à `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.

UserErrors

Demandes adressées à DynamoDB ou à Amazon DynamoDB Streams qui génèrent un code d'état HTTP 400 durant la période spécifiée. Une erreur HTTP 400 indique généralement une erreur côté client, comme une combinaison de paramètres non valide, une tentative de mise à jour d'une table inexistante ou une signature de demande incorrecte.

Voici quelques exemples d'exceptions qui enregistrent les métriques liées à `UserErrors` :

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Tous ces événements sont reflétés dans la métrique `UserErrors`, à l'exception des événements suivants :

- ProvisionedThroughputExceededException— Voir la `ThrottledRequests` métrique dans cette section.
- ConditionalCheckFailedException— Voir la `ConditionalCheckFailedRequests` métrique dans cette section.

`UserErrors`représente l'ensemble des erreurs HTTP 400 pour les demandes DynamoDB ou Amazon DynamoDB Streams pour la région actuelle et le compte courant.AWSAWS

Unités `Count`:

Statistiques valides :

- `Sum`
- `SampleCount`

WriteAccountLimitThrottleEvents

Nombre de demandes d'écriture limitées en raison des limites du compte.

Unités : `Count`

Dimensions : `TableName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

WriteKeyRangeThroughputThrottleEvents

Nombre de demandes d'écriture limitées en raison des limites de partition.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

WriteMaxOnDemandThroughputThrottleEvents

Nombre de demandes d'écriture limitées en raison du débit maximal à la demande.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.

- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

WriteProvisionedThroughputThrottleEvents

Nombre de demandes d'écriture limitées en raison de limites de débit provisionné.

Unités : `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistiques valides :

- `Sum` – Nombre total d'événements limités.
- `SampleCount` – Nombre d'occurrences de limitation.
- `Minimum` – Nombre minimum d'événements limités dans un exemple donné.
- `Maximum` – Nombre maximum d'événements limités dans un exemple donné.

WriteThrottleEvents

Demandes adressées à DynamoDB qui dépassent le nombre d'unités de capacité d'écriture provisionnée pour une table ou un index secondaire global.

Une seule demande peut entraîner plusieurs événements. Par exemple, une demande `PutItem` sur une table avec trois index secondaires globaux entraîne quatre événements : une écriture dans la table et trois écritures dans les index. Pour chaque événement, la métrique `WriteThrottleEvents` est incrémentée d'une unité si l'événement est limité. Pour les demandes `PutItem` uniques, si l'un des événements est limité, la valeur `ThrottledRequests` est également incrémentée d'une unité. Concernant l'opération `BatchWriteItem`, la métrique `ThrottledRequests` pour l'opération `BatchWriteItem` entière n'est incrémentée que si tous les événements `PutItem` ou `DeleteItem` sont limités.

La dimension `TableName` renvoie la valeur `WriteThrottleEvents` pour la table, mais pas pour les index secondaires globaux. Pour afficher la valeur `WriteThrottleEvents` pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.

Unités : `Count`

Dimensions : `TableName, GlobalSecondaryIndexName`

Statistiques valides :

- `Sum`
- `SampleCount`

Métriques d’utilisation

Les métriques d'utilisation vous CloudWatch permettent de gérer l'utilisation de manière proactive en visualisant les métriques dans la CloudWatch console, en créant des tableaux de bord personnalisés, en détectant les changements d'activité grâce à la détection des CloudWatch anomalies et en configurant des alarmes qui vous alertent lorsque l'utilisation approche un seuil.

DynamoDB intègre également ces métriques d’utilisation avec Service Quotas. Vous pouvez l'utiliser CloudWatch pour gérer l'utilisation de vos quotas de service par votre compte. Pour de plus amples informations, consultez Visualisation de vos quotas de service et définition des alarmes.

Liste des métriques d’utilisation disponibles

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

Nombre d’unités de capacité d’écriture provisionnées pour une table ou un index secondaire global.

Unités `Count`:

Statistiques valides :

- `Minimum` – Nombre d’unités de capacité d’écriture provisionnées le plus faible pendant une période.
- `Maximum` – Nombre d’unités de capacité d’écriture provisionnéed le plus élevé pendant une période.
- `Average` – Nombre moyen d’unités de capacité d’écriture provisionnées au cours d’une période.

La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité d’écriture provisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

AccountProvisionedReadCapacityUnits

Nombre d'unités de capacité de lecture provisionnées pour une table ou un index secondaire global.

Unités `Count`:

Statistiques valides :

- `Minimum` – Nombre d'unités de capacité de lecture provisionnées le plus faible pendant une période.
- `Maximum` – Nombre d'unités de capacité de lecture provisionnées le plus élevé pendant une période.
- `Average` – Nombre moyen d'unités de capacité de lecture provisionnées au cours d'une période.

La métrique est publiée à intervalles de cinq minutes. Par conséquent, si vous ajustez rapidement les unités de capacité de lecture provisionnée, il se peut que cette statistique ne reflète pas la moyenne réelle.

TableCount

Nombre de tables actives d'un compte.

Unités `Count`:

Statistiques valides :

- `Minimum` – Nombre de tables le plus faible au cours d'une période.
- `Maximum` – Nombre de tables le plus élevé pendant une période.
- `Average` – Nombre de tables moyen pendant une période.

Comprendre les métriques et dimensions pour DynamoDB

Les métriques associées à DynamoDB sont qualifiées par les valeurs du compte, du nom de table, du nom d'index secondaire ou de l'opération. Vous pouvez utiliser la CloudWatch console pour récupérer des données DynamoDB selon l'une des dimensions du tableau ci-dessous.

Liste des dimensions disponibles

- DelegatedOperation

- GlobalSecondaryIndexName
- Opération
- OperationType
- Verb
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

Cette dimension limite les données aux opérations que DynamoDB effectue pour vous. Les opérations capturées sont les suivantes :

- récupération de données de modification pour Kinesis Data Streams.

GlobalSecondaryIndexName

Cette dimension limite les données à un index secondaire global sur une table. Si vous spécifiez `GlobalSecondaryIndexName`, vous devez également spécifier `TableName`.

Opération

Cette dimension limite les données à l'une des opérations DynamoDB suivantes :

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`

- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

En outre, vous pouvez limiter les données à l'opération d'Amazon DynamoDB Streams suivante :

- `GetRecords`

OperationType

Cette dimension limite les données à l'un des types d'opérations suivants :

- `Read`
- `Write`

Cette dimension est émise pour des demandes `ExecuteTransaction` et `BatchExecuteStatement`.

Verb

Cette dimension limite les données à l’un des verbes DynamoDB PartiQL suivants :

- Insert : `PartiQLInsert`
- Select : `PartiQLSelect`
- Update : `PartiQLUpdate`
- Delete : `PartiQLDelete`

Cette dimension est émise pour l’opération `ExecuteStatement`.

ReceivingRegion

Cette dimension limite les données à une AWS région spécifique. Elle est utilisée avec des métriques provenant de tables de réplique au sein d'une table globale DynamoDB.

StreamLabel

Cette dimension limite les données à une étiquette de flux spécifique. Elle est utilisée avec des métriques provenant d'opérations `GetRecords` d'Amazon DynamoDB Streams.

TableName

Cette dimension limite les données à une table spécifique. Cette valeur peut être n'importe quel nom de table de la région actuelle et du AWS compte courant.

Création d' CloudWatch alarmes dans DynamoDB

Une CloudWatch alarme surveille une seule métrique sur une période spécifiée et exécute une ou plusieurs actions spécifiées, en fonction de la valeur de la métrique par rapport à un seuil au fil du temps. L'action est une notification envoyée à une rubrique Amazon SNS ou à une politique Auto Scaling. Vous pouvez également ajouter des alarmes aux tableaux de bord afin de surveiller et de recevoir des alertes concernant vos AWS ressources et applications dans plusieurs régions. Le nombre d'alarmes que vous pouvez créer est illimité. CloudWatch les alarmes n'appellent pas d'actions simplement parce qu'elles sont dans un état particulier ; l'état doit avoir changé et être maintenu pendant un certain nombre de périodes. Pour obtenir la liste des alarmes DynamoDB recommandées, consultez Alarmes recommandées.

> **Note**
>
> Vous devez spécifier toutes les dimensions requises lors de la création de votre CloudWatch alarme, car les mesures relatives à une dimension manquante ne CloudWatch seront pas agrégées. La création CloudWatch d'une alarme avec une dimension manquante ne provoquera pas d'erreur lors de la création de l'alarme.

Supposons que vous disposiez d'une table provisionnée avec cinq unités de capacité de lecture. Vous souhaitez être averti avant de consommer la totalité de la capacité de lecture allouée. Vous décidez donc de créer une CloudWatch alarme pour être averti lorsque la capacité consommée atteint 80 % de ce que vous avez provisionné pour la table. Vous pouvez créer des alarmes dans la CloudWatch console ou à l'aide du AWS CLI.

Création d'une alarme dans la CloudWatch console

Pour créer une alarme dans la CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Dans le panneau de navigation, choisissez Alarms (alertes), All alarms (Toutes les alertes).

3. Choisissez Create alarm (Créer une alerte).
4. Recherchez la rangée contenant le tableau que vous souhaitez surveiller et **`ConsumeReadCapacityUnits`** dans la colonne Nom de la métrique. Cochez la case en regard de cette ligne, puis choisissez Sélectionner la métrique.
5. Sous Spécifier les métriques et les conditions, dans le champ Statistique, sélectionnez Somme. Choisissez une période de 1 minute.
6. Sous Conditions, spécifiez les éléments suivants :
   a. Pour Threshold type (Type de seuil), choisissez Static (Statique).
   b. Pour Chaque fois que **`ConsumedReadCapacityUnits`** est, choisissez Supérieur à/Égal à et spécifiez le seuil 240.
7. Choisissez Suivant.
8. Sous Notification, choisissez **`In alarm`** et sélectionnez une rubrique SNS à notifier lorsque l'alerte est en état `ALARM`.
9. Lorsque vous avez terminé, choisissez Suivant.
10. Saisissez le nom et la description de l'alarme, puis choisissez Suivant.
11. Dans Prévisualiser et créer, confirmez que les informations et les conditions sont telles que vous les voulez, puis choisissez Créer une alarme.

Création d'une alarme dans le AWS CLI

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

Testez l'alarme.

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

Plus d'AWS CLI exemples

La procédure suivante décrit comment vous êtes averti si vos demandes dépassent les quotas de débit provisionnés pour une table.

1. Créez une rubrique Amazon SNS. `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput` Pour plus d'informations, consultez Configuration d'Amazon Simple Notification Service.
2. Créez l'alerte.

   ```
   aws cloudwatch put-metric-alarm \
           -\-alarm-name ReadCapacityUnitsLimitAlarm \
           -\-alarm-description "Alarm when read capacity reaches 80% of my
    provisioned read capacity" \
           -\-namespace AWS/DynamoDB \
           -\-metric-name ConsumedReadCapacityUnits \
           -\-dimensions Name=TableName,Value=myTable \
           -\-statistic Sum \
           -\-threshold 240 \
           -\-comparison-operator GreaterThanOrEqualToThreshold \
           -\-period 60 \
           -\-evaluation-periods 1 \
           -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
   ```
3. Testez l'alarme.

   ```
   aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
   state-reason "initializing" --state-value OK
   ```

   ```
   aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
   state-reason "initializing" --state-value ALARM
   ```

La procédure suivante décrit comment vous êtes averti en cas d'erreur système.

1. Créez une rubrique Amazon SNS. `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors` Pour plus d'informations, consultez Configuration d'Amazon Simple Notification Service.
2. Créez l'alerte.

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Testez l'alarme.

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail

DynamoDB est intégré avec AWS CloudTrail, service qui enregistre les actions accomplies par un utilisateur, un rôle ou un service AWS dans DynamoDB. CloudTrail récupère tous les appels d'API pour DynamoDB en tant qu'événements. Les appels capturés incluent les appels de la console DynamoDB et les appels de code aux opérations d'API DynamoDB, en utilisant à la fois PartiQL et l'API classique. Si vous créez un journal de suivi, vous pouvez activer la livraison continue des événements CloudTrail dans un compartiment Amazon S3, y compris les événements

pour DynamoDB. Si vous ne configurez pas de journal d'activité, vous pouvez toujours afficher les événements les plus récents dans la console CloudTrail dans Event history (Historique des événements). Avec les informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été adressée à DynamoDB, l'adresse IP à partir de laquelle la demande a été effectuée, l'auteur et la date de la demande, ainsi que d'autres détails.

Pour une surveillance et des alertes robustes, vous pouvez également intégrer des événements CloudTrail avec Amazon CloudWatch Logs. Pour améliorer votre analyse de l'activité de service DynamoDB et identifier les changements dans les activités d'un compte AWS, vous pouvez interroger les journaux AWS CloudTrail à l'aide d'Amazon Athena. Par exemple, vous pouvez utiliser des requêtes pour identifier des tendances et isoler davantage l'activité par attributs, comme l'adresse IP source ou l'utilisateur.

Pour en savoir plus sur CloudTrail, y compris la façon de le configurer et de l'activer, consultez le AWS CloudTrail Guide de l'utilisateur.

Rubriques

- Informations de DynamoDB dans CloudTrail
- Présentation des entrées de fichier journal DynamoDB

Informations de DynamoDB dans CloudTrail

CloudTrail est activé dans votre compte AWS lors de la création de ce dernier. Quand une activité d'événement prise en charge a lieu dans DynamoDB, elle est enregistrée dans un événement CloudTrail avec d'autres événements de service AWS, dans Event history (Historique des événements). Vous pouvez afficher, rechercher et télécharger les événements récents dans votre compte AWS. Pour plus d'informations, consultez Utilisation de l'historique des événements CloudTrail.

Pour un registre continu des événements dans votre compte AWS, y compris des événements pour DynamoDB, créez un journal d'activité. Un journal de suivi permet à CloudTrail de livrer des fichiers journaux à compartiment Amazon S3. Par défaut, lorsque vous créez un journal de suivi dans la console, il s'applique à toutes les régions AWS. Le journal de suivi consigne les événements de toutes les Régions dans la partition AWS et livre les fichiers journaux dans le compartiment Amazon S3 de votre choix. En outre, vous pouvez configurer d'autres services AWS pour analyser et agir sur les données d'événements collectées dans les journaux CloudTrail. Pour plus d'informations, consultez les ressources suivantes :

- Présentation de la création d'un journal de suivi
- Intégrations et services pris en charge par CloudTrail
- Configuration des notifications Amazon SNS pour CloudTrail
- Réception de fichiers journaux CloudTrail de plusieurs régions et Réception de fichiers journaux CloudTrail de plusieurs comptes

Contrôler les événements de plan dans CloudTrail

Les actions d'API suivantes sont journalisées par défaut en tant qu'événements dans des fichiers CloudTrail :

Amazon DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup
- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable

- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity
- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB Streams

- DescribeStream
- ListStreams

DynamoDB Accelerator (DAX)

- CreateCluster
- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups
- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource

- UntagResource
- UpdateCluster
- UpdateParameterGroup
- UpdateSubnetGroup

Événements de plan de données DynamoDB dans CloudTrail

Pour activer la journalisation des actions d'API suivantes dans des fichiers CloudTrail, vous devez activer la journalisation de l'activité de l'API de plan de données dans CloudTrail. Pour plus d'informations, consultez Consignation d'événements de données pour les journaux d'activité.

Les événements de plan de données peuvent être filtrés par type de ressource pour un contrôle granulaire des appels d'API DynamoDB que vous souhaitez journaliser et payer de manière sélective dans CloudTrail. Par exemple, en spécifiant `AWS::DynamoDB::Stream` comme type de ressource, vous pouvez journaliser uniquement les appels aux API de flux DynamoDB. Pour les tables ayant les flux activés, le champ de ressource dans l'événement de plan de données contient à la fois `AWS::DynamoDB::Stream` et `AWS::DynamoDB::Table`. Si vous spécifiez `AWS::DynamoDB::Table` comme type de ressource, les événements de table DynamoDB et les événements de flux DynamoDB sont journalisés par défaut. Vous pouvez ajouter un filtre supplémentaire pour exclure les événements de flux si vous ne voulez pas que ceux-ci soient journalisés. Pour plus d'informations, consultez DataResource dans la Référence de l'API AWS CloudTrail.

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Interrogation
- Analyser

- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> Les actions du plan de données TTL DynamoDB ne sont pas consignées par CloudTrail

DynamoDB Streams

- GetRecords
- GetShardIterator

Présentation des entrées de fichier journal DynamoDB

Un journal de suivi est une configuration qui permet d'envoyer des événements sous forme de fichiers journaux à un compartiment Amazon S3 de votre choix. Les fichiers journaux CloudTrail peuvent contenir une ou plusieurs entrées. Un événement représente une demande unique provenant de n'importe quelle source et comprend des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec les informations d'identification utilisateur racine ou .
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la demande a été effectuée par un autre service AWS.

> **Note**
>
> Les valeurs d'attribut autres que de clé sont effacées dans les journaux CloudTrail d'actions utilisant l'API PartiQL et n'apparaissent pas dans les journaux d'actions utilisant l'API classique.

Pour plus d'informations, consultez l'élément userIdentity CloudTrail.

Les exemples suivants illustrent les journaux CloudTrail de ces types d'événements :

Amazon DynamoDB

- UpdateTable
- DeleteTable
- CreateCluster
- PutItem (Réussite)
- UpdateItem (Échec)
- TransactWriteItems (Réussite)
- TransactWriteItems (Avec TransactionCanceledException)
- ExecuteStatement
- BatchExecuteStatement

DynamoDB Streams

- GetRecords

UpdateTable


```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
```

```
                    "type": "Role",
                    "principalId": "AKIAI44QH8DHBEXAMPLE",
                    "arn": "arn:aws:iam::444455556666:role/admin-role",
                    "accountId": "444455556666",
                    "userName": "bob"
                }
            }
        },
        "eventTime": "2015-05-04T02:14:52Z",
        "eventSource": "dynamodb.amazonaws.com",
        "eventName": "UpdateTable",
        "awsRegion": "us-west-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "console.aws.amazon.com",
        "requestParameters": {
            "provisionedThroughput": {
                "writeCapacityUnits": 25,
                "readCapacityUnits": 25
            }
        },
        "responseElements": {
            "tableDescription": {
                "tableName": "Music",
                "attributeDefinitions": [
                    {
                        "attributeType": "S",
                        "attributeName": "Artist"
                    },
                    {
                        "attributeType": "S",
                        "attributeName": "SongTitle"
                    }
                ],
                "itemCount": 0,
                "provisionedThroughput": {
                    "writeCapacityUnits": 10,
                    "numberOfDecreasesToday": 0,
                    "readCapacityUnits": 10,
                    "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
                },
                "creationDateTime": "May 3, 2015 11:34:14 PM",
                "keySchema": [
                    {
                        "attributeName": "Artist",
```

```
                            "keyType": "HASH"
                        },
                        {
                            "attributeName": "SongTitle",
                            "keyType": "RANGE"
                        }
                    ],
                    "tableStatus": "UPDATING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
```

```
                    }
                }
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

CreateCluster

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
```

```
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
```

```
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

PutItem (Réussite)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
```

```
                    }
                }
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

UpdateItem (Échec)


```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
```

```
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

`TransactWriteItems` (Réussite)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
```

```
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
```

```
                    "ConditionExpression": "attribute_not_exists(Rating)",
                    "ExpressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Delete",
                    "TableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
                    "ExpressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                }
            ],
            "returnConsumedCapacity": "TOTAL",
            "returnItemCollectionMetrics": "SIZE"
        },
        "responseElements": null,
        "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
        "readOnly": false,
        "resources": [
            {
                "accountId": "111122223333",
                "type": "AWS::DynamoDB::Table",
```

```
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

TransactWriteItems (Avec **TransactionCanceledException**)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
```

```
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
```

```
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
                    "ExpressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                }
            ],
            "returnConsumedCapacity": "TOTAL",
            "returnItemCollectionMetrics": "SIZE"
        },
        "responseElements": null,
        "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
        "readOnly": false,
        "resources": [
            {
                "accountId": "111122223333",
                "type": "AWS::DynamoDB::Table",
                "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
            }
        ],
        "eventType": "AwsApiCall",
        "apiVersion": "2012-08-10",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data"
    }
]
```

```
}
```

ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
 SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
```

```
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

BatchExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
```

```
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE
 Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
                        "statement": "INSERT INTO Music VALUE {'Artist' :
 ***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

GetRecords

```
{
    "Records": [
        {
```

```
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
```

```
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```


Analyse de l'accès aux données à l'aide des informations des CloudWatch contributeurs pour DynamoDB

Amazon CloudWatch Contributor Insights pour Amazon DynamoDB est un outil de diagnostic permettant d'identifier en un coup d'œil les clés les plus fréquemment consultées et les plus limitées de votre table ou de votre index. Cet outil utilise les informations des CloudWatch contributeurs.

En activant CloudWatch Contributor Insights pour DynamoDB sur une table ou un index secondaire global, vous pouvez afficher les éléments les plus consultés et les plus limités de ces ressources.

> **Note**
>
> CloudWatch des frais s'appliquent pour Contributor Insights for DynamoDB. Pour plus d'informations sur les tarifs, consultez les CloudWatchtarifs Amazon.

Rubriques

- CloudWatch informations fournies par les contributeurs pour DynamoDB : comment cela fonctionne

- Mise en route avec CloudWatch Contributor Insights for DynamoDB
- Utilisation d'IAM avec les informations des CloudWatch contributeurs pour DynamoDB

CloudWatch informations fournies par les contributeurs pour DynamoDB : comment cela fonctionne

Amazon DynamoDB s'intègre CloudWatch à Contributor Insights pour fournir des informations sur les éléments les plus consultés et les plus limités dans un tableau ou un index secondaire global. DynamoDB vous fournit ces informations CloudWatch via les règles, les rapports et les graphiques des données des rapports de Contributor Insights.

CloudWatch Contributor Insights for DynamoDB est conçu pour n'avoir aucun impact sur les performances de votre table DynamoDB.

Pour plus d'informations sur CloudWatch Contributor Insights, consultez la section Utilisation de Contributor Insights pour analyser les données à cardinalité élevée dans le guide de CloudWatch l'utilisateur Amazon.

Les sections suivantes décrivent les concepts fondamentaux et le comportement de CloudWatch Contributor Insights for DynamoDB.

Rubriques

- CloudWatch modes d'analyse des contributeurs pour DynamoDB
- CloudWatch informations des contributeurs sur les règles DynamoDB
- Comprendre les informations des CloudWatch contributeurs pour les graphes DynamoDB
- Interactions avec d’autres fonctions DynamoDB
- CloudWatch informations sur les contributeurs pour la facturation DynamoDB

CloudWatch modes d'analyse des contributeurs pour DynamoDB

CloudWatch Contributor Insights for DynamoDB propose deux modes distincts pour répondre aux différents besoins de surveillance.

Mode clés limitées

Ce mode se concentre exclusivement sur les demandes limitées en traitant uniquement les événements lorsque la limitation se produit. Il fournit des informations sur les problèmes de

performance sans avoir à suivre tous les modèles d'accès. Dans ce mode, DynamoDB suit uniquement les éléments suivants :

- éléments les plus limités : éléments qui subissent le plus de limitations.

Ce mode est idéal lorsque :

- votre principale préoccupation est d'identifier et de résoudre les problèmes de limitation ;
- vous souhaitez que Contributor Insights reste activé en permanence pour les alertes de limitation en temps réel ;
- vous souhaitez une approche optimisée en termes de coûts pour surveiller les problèmes de limitation.

> **Note**
>
> Le mode clés limitées ne traite les événements qu'en cas de limitation, ce qui permet une surveillance continue rentable. Cette approche ciblée vous permet de laisser la fonctionnalité activée en permanence avec un impact financier minimal, tout en offrant une visibilité immédiate sur les problèmes de limitation au fur et à mesure qu'ils surviennent.

Si votre table n'est pas limitée, vous ne verrez aucune donnée dans les graphiques de Contributor Insights, ce qui indique de bonnes performances. Lorsqu'une limitation est détectée, les graphiques générés vous aident à identifier les modèles d'accès spécifiques à l'origine de problèmes de performances. Ces informations peuvent vous aider à mettre en œuvre des stratégies pour remédier aux modèles d'accès non uniformes.

Pour des stratégies de surveillance complètes, vous pouvez intégrer ces informations de régulation à d'autres CloudWatch indicateurs afin de créer des tableaux de bord unifiés qui mettent en corrélation les événements de régulation avec les performances globales des tables.

Mode clés consultées et limitées

Ce mode fournit une surveillance complète des éléments consultés et limités. Dans ce mode, DynamoDB suit les éléments suivants :

- éléments ayant le plus grand nombre d'accès : éléments qui consomment le plus de capacité de lecture et d'écriture ;

- éléments les plus limités : éléments qui subissent le plus de limitations.

Ce mode est idéal lorsque vous avez besoin d'une visibilité complète sur les modèles d'accès de votre table et que vous souhaitez comprendre à la fois les éléments à fort trafic et les problèmes de limitation.

Basculement entre les modes

Vous pouvez passer d'un mode à l'autre à tout moment à l'aide de la console AWS CLI DynamoDB, ou. APIs Lorsque vous changez de mode :

- Les CloudWatch règles existantes sont mises à jour pour correspondre au nouveau mode
- Les CloudWatch règles relatives aux touches limitées restent intactes, ce qui permet de conserver vos données historiques continues pour les mesures de régulation :
  - lorsque vous passez du mode clés limitées au mode clés consultées et limitées, les règles relatives aux clés limitées existantes sont préservées et de nouvelles règles de clés consultées sont créées ;
  - lorsque vous passez du mode clés consultées et limitées au mode clés limitées, seules les règles relatives aux clés limitées sont préservées et les règles de clés consultées sont supprimées.
- La facturation s'ajuste immédiatement pour refléter le traitement des événements dans le nouveau mode.

CloudWatch informations des contributeurs sur les règles DynamoDB

Lorsque vous activez CloudWatch Contributor Insights pour DynamoDB sur une table ou un index secondaire global, DynamoDB crée des règles en votre nom en fonction du mode sélectionné.

> **Note**
>
> Lorsque vous activez Contributor Insights sur votre table DynamoDB, vous êtes soumis aux limites des règles de Contributor Insights. Pour plus d'informations, consultez Quotas de service CloudWatch .

Règles pour le mode clés consultées et limitées

En mode clés consultées et limitées, DynamoDB crée les règles suivantes :

- Éléments les plus consultés (clés de partition) : identifie les clés de partition des éléments les plus consultés de votre table ou de votre index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`

- Clés les plus limitées (clés de partition) : identifie les clés de partition des éléments les plus limités de votre table ou de votre index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Si votre table ou votre index secondaire global comporte des clés de tri, DynamoDB crée également les règles suivantes spécifiques aux clés de tri :

- Clés les plus utilisées (clés de partition et de tri) – Identifie les clés de partition et de tri des éléments les plus consultés dans votre table ou index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`

- Clés les plus limitées (clés de partition et de tri) – Identifie les clés de partition et de tri des éléments les plus limités dans votre table ou index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Règles relatives au mode clés limitées

En mode clés limitées, DynamoDB crée uniquement les règles liées à la limitation :

- Clés les plus limitées (clé de partition) – Identifie les clés de partition des éléments les plus limités dans votre table ou index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Si votre table ou votre index secondaire global comporte des clés de tri, DynamoDB crée également :

- Clés les plus limitées (clés de partition et de tri) – Identifie les clés de partition et de tri des éléments les plus limités dans votre table ou index secondaire global.

  CloudWatch format du nom de règle : `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Cette approche ciblée réduit le nombre de règles actives et le volume d'événements traités afin de mieux diagnostiquer vos événements de limitation.

> **Note**
>
> - Lorsque vous utilisez la CloudWatch console ou APIs que vous consultez CloudWatch Contributor Insights for DynamoDB, seules les règles correspondant au mode sélectionné s'affichent.
> - Vous ne pouvez pas utiliser la CloudWatch console ni APIs modifier ou supprimer directement les règles créées par CloudWatch Contributor Insights pour DynamoDB. La désactivation de CloudWatch Contributor Insights pour DynamoDB sur une table ou un index secondaire global supprime automatiquement les règles créées pour cette table ou cet index secondaire global.
> - Lorsque vous utilisez l'GetInsightRuleReportopération avec des règles CloudWatch Contributor Insights créées par DynamoDB, `MaxContributorValue` uniquement `Maximum` et que vous renvoyez des statistiques utiles. Les autres statistiques de cette liste ne renvoient pas de valeurs intéressantes.
> - CloudWatch Contributor Insights for DynamoDB est limité à 25 contributeurs. Si vous demandez plus de 25 contributeurs, une erreur s'affichera.

Vous pouvez créer des CloudWatch alarmes à l'aide des CloudWatch règles Contributor Insights for DynamoDB. Cela vous permet d'être prévenu lorsqu'un élément dépasse ou atteint un seuil spécifique pour `ConsumedThroughputUnits` ou `ThrottleCount`. Pour plus d'informations, consultez Définition d'une alarme sur les données de métrique de Contributor Insights.

Comprendre les informations des CloudWatch contributeurs pour les graphes DynamoDB

CloudWatch Contributor Insights for DynamoDB affiche différents types de graphiques à la fois sur DynamoDB et sur les consoles en fonction du mode CloudWatch sélectionné.

Disponibilité des graphiques par mode

Les graphiques affichés dépendent du mode Contributor Insights que vous avez sélectionné.

- Le mode clés consultées et limitées affiche à la fois les graphiques Éléments les plus consultés et Éléments les plus limités.
- Le mode clés limitées affiche uniquement les graphiques Éléments les plus limités.

Éléments les plus consultés

Ce graphique n'est disponible qu'en mode clés consultées et limitées. Utilisez ce graphique pour identifier les éléments les plus consultés dans la table ou l'index secondaire global. Le graphique affiche `ConsumedThroughputUnits` sur l'axe y et le temps sur l'axe x. Chacune des clés N principales est affichée avec sa couleur et une légende sous l'axe x.

DynamoDB mesure la fréquence d'accès des clés en utilisant `ConsumedThroughputUnits`, qui mesure le trafic de lecture et d'écriture. `ConsumedThroughputUnits` est défini comme ce qui suit :

- Approvisionné – (3 x unités de capacité d'écriture consommées) + unités de capacité de lecture consommées
- À la demande – (3 x unités de demande d'écriture) + unités de demande d'écriture

Sur la console DynamoDB, chaque point de données du graphique représente la valeur `ConsumedThroughputUnits` maximum sur une période d'une minute. Par exemple, une valeur de graphique de 180 000 `ConsumedThroughputUnits` indique que l'élément a été consulté en continu au débit maximum par élément de 1 000 unités de demande d'écriture ou de 3 000 unités de demande de lecture pendant 60 secondes au cours de cette période d'une minute (3 000 x 60 secondes). En d'autres termes, les valeurs du graphique représentent la minute de trafic maximal pendant chaque période d'une minute. Vous pouvez modifier la granularité temporelle de la `ConsumedThroughputUnits` métrique (par exemple, pour afficher les métriques de 5 minutes au lieu d'une minute) sur la CloudWatch console.

Si vous voyez plusieurs lignes étroitement regroupées en cluster sans aberrations évidentes, cela indique que votre charge de travail est relativement équilibrée pour les éléments au cours de la période donnée. Si vous voyez des points isolés dans le graphique plutôt que des lignes reliées, cela indique qu'un élément a été consulté fréquemment uniquement pendant une courte période.

Si votre table ou votre index secondaire global comporte des clés de tri, DynamoDB crée deux graphiques : l'un pour les clés de partition les plus consultées et l'autre pour les paires partition+clés de tri les plus consultées. Vous pouvez voir le trafic au niveau des clés de partition dans le graphique contenant uniquement les clés de partition. Vous pouvez voir le trafic au niveau de l'article dans les graphiques des partitions et des clés de tri.

Éléments les plus limités

Ce graphique est disponible dans les deux modes. Utilisez ce graphique pour identifier les éléments les plus limités dans la table ou l'index secondaire global. Le graphique affiche `ThrottleCount` sur l'axe y et le temps sur l'axe x. Chacune des clés N principales est affichée avec sa couleur et une légende sous l'axe x.

DynamoDB mesure la fréquence de limitation à l'aide de `ThrottleCount`, c'est-à-dire le nombre d'erreurs `ProvisionedThroughputExceededException`, `ThrottlingException` et `RequestLimitExceeded`.

La limitation des écritures résultant d'une capacité d'écriture insuffisante pour un index secondaire global n'est pas mesurée. Vous pouvez utiliser le graphique Éléments les plus utilisés de l'index secondaire global pour identifier les modèles d'accès non équilibrés susceptibles de provoquer une limitation d'écriture. Pour plus d'informations, consultez Considérations sur le débit provisionné pour les index secondaires globaux.

Sur la console DynamoDB, chaque point du graphique représente le nombre d'événements de limitation sur une période d'une minute.

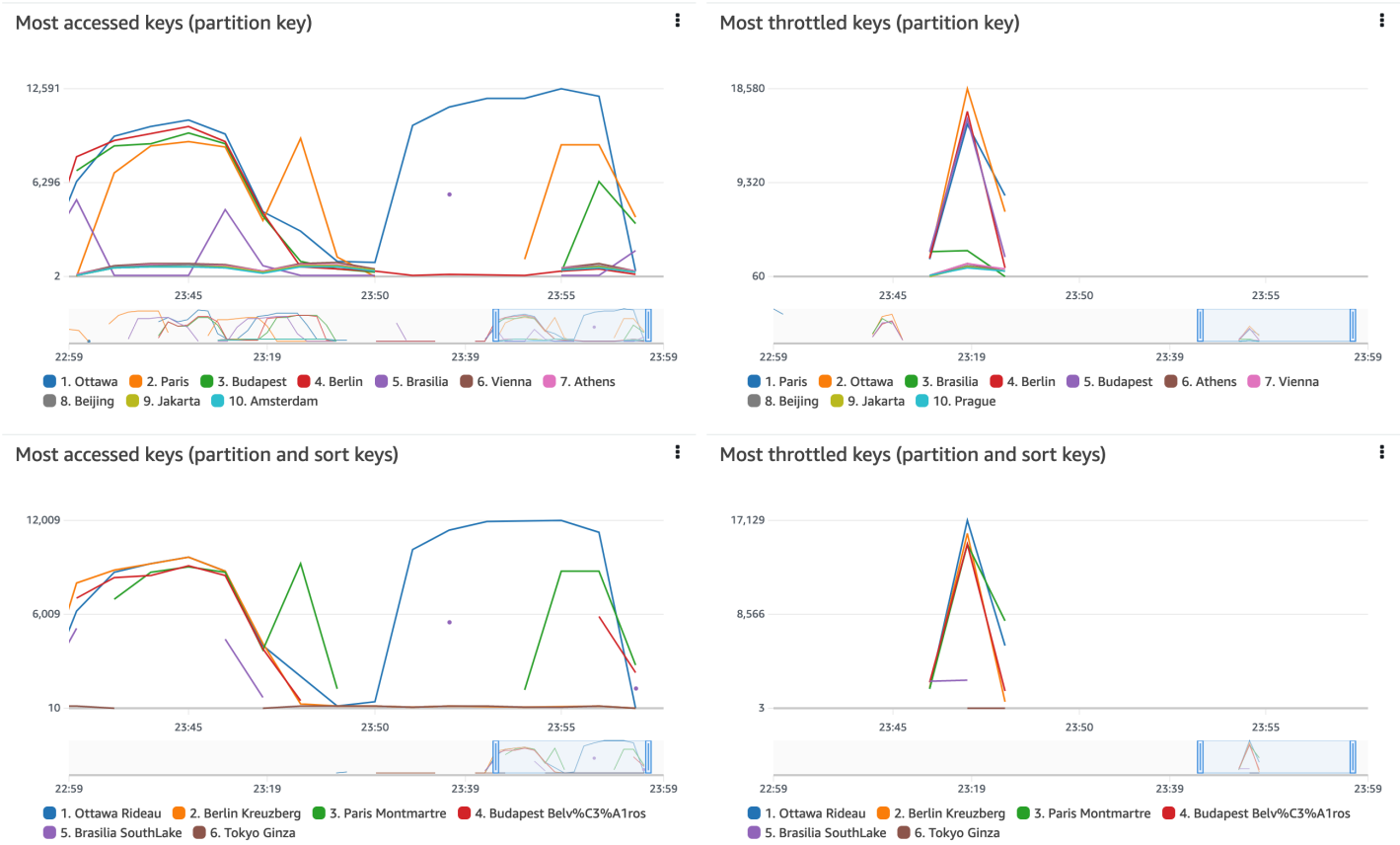
Si vous ne voyez pas de données dans ce graphique, cela indique que vos demandes ne sont pas limitées. Si vous voyez des points isolés dans le graphique plutôt que des lignes reliées, cela indique qu'un élément a été fréquemment limité pendant une courte période.

Si votre table ou votre index secondaire global comporte des clés de tri, DynamoDB crée deux graphiques : l'un pour la plupart des clés de partition limitées et l'autre pour les paires partitions limitées et clés de tri les plus limitées. Vous pouvez voir le nombre d'accélérateurs au niveau des clés de partition dans le graphique des clés de partition uniquement, et le nombre d'accélérateurs au niveau des éléments dans le graphique des clés de tri des partitions et des clés de tri.

> **Note**
>
> En mode clés limitées, il s'agit du seul type de graphique que vous verrez. L'absence de données dans ces graphiques indique une bonne performance des tables sans aucune limitation.

Exemples de rapports

L'exemple suivant montre les rapports générés pour une table contenant à la fois des clés de partition et des clés de tri en mode touches accessibles et limitées. En mode clés limitées, vous ne voyez que la partie de ce rapport relative à la limitation.



Interactions avec d'autres fonctions DynamoDB

Les sections suivantes décrivent le comportement et l'interaction de CloudWatch Contributor Insights for DynamoDB avec plusieurs autres fonctionnalités de DynamoDB. Sauf indication contraire, ces comportements s'appliquent aux deux modes.

Tables globales

CloudWatch Contributor Insights for DynamoDB surveille les répliques de tables globales en tant que tables distinctes. Les graphiques Contributor Insights d'une réplique dans une AWS région peuvent ne pas présenter les mêmes modèles que ceux d'une autre région. En effet, les données d'écriture sont répliquées sur tous les réplicas dans une table globale, mais chaque réplica peut servir le trafic en lecture limité à la région.

Chaque réplica peut être configuré indépendamment avec un mode Contributor Insights différent. Par exemple, vous pouvez utiliser le mode clés consultées et limitées dans votre région principale pour une surveillance complète, tandis que le mode clés limitées dans les régions secondaires permet de conserver une visibilité sur les problèmes de performance.

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights for DynamoDB n'affiche pas les réponses du cache DAX. Il affiche uniquement les réponses à l'accès à une table ou à un index secondaire global.

> **Note**
>
> CloudWatch DynamoDB Contributor Insights ne prend pas en charge les requêtes PartiQL.

Chiffrement au repos

CloudWatch Contributor Insights for DynamoDB n'a aucune incidence sur le fonctionnement du chiffrement dans DynamoDB. Les données de clé primaire publiées dans CloudWatch sont cryptées avec le Clé détenue par AWS. Cependant, DynamoDB prend également en charge la clé et une clé gérée Clé gérée par AWS par le client.

CloudWatch Contributor Insights for DynamoDB affiche les clés de partition et les clés de tri (le cas échéant) des éléments fréquemment consultés et limités. Bien que CloudWatch Contributor Insights fonctionne avec des tables DynamoDB chiffrées, il est important de noter qu'il utilise son propre contexte de chiffrement appartenant à Amazon, qui est distinct du chiffrement configuré de la table.

Si la clé primaire de votre table DynamoDB contient des informations sensibles et que les politiques de sécurité de votre organisation exigent un contrôle total des processus de chiffrement, l' CloudWatch activation de Contributor Insights peut ne pas être appropriée.

Contrôle précis des accès

CloudWatch Contributor Insights for DynamoDB ne fonctionne pas différemment pour les tables dotées d'un contrôle d'accès détaillé (FGAC). En d'autres termes, tout utilisateur disposant des CloudWatch autorisations appropriées peut consulter les clés primaires protégées par le FGAC dans les graphiques de CloudWatch Contributor Insights.

Si la clé primaire de la table contient des données protégées par le FGAC sur lesquelles vous ne souhaitez pas publier CloudWatch, vous ne devez pas activer CloudWatch Contributor Insights for DynamoDB pour cette table.

Contrôle d’accès

Vous contrôlez l'accès à CloudWatch Contributor Insights for DynamoDB à Gestion des identités et des accès AWS l'aide de (IAM) en limitant les autorisations du plan de contrôle DynamoDB et les autorisations du plan de données. CloudWatch Pour plus d'informations, voir Utilisation d'IAM avec CloudWatch Contributor Insights pour DynamoDB.

CloudWatch informations sur les contributeurs pour la facturation DynamoDB

Les frais associés à CloudWatch Contributor Insights for DynamoDB figurent dans CloudWatchla section de votre facture mensuelle. Ces frais sont calculés en fonction du nombre d’événements de DynamoDB traités et du mode sélectionné.

Facturation par mode

Les deux modes de Contributor Insights présentent des caractéristiques de facturation différentes.

- Facturation en mode clés consultées et limitées : dans ce mode, chaque élément écrit ou lu via une opération de plan de données représente un événement, que la demande aboutisse ou soit limitée. Si une table ou un index secondaire global inclut des clés de tri, chaque élément lu ou écrit représente deux événements. Cela est dû au fait que DynamoDB identifie les principaux contributeurs à partir de séries chronologiques distinctes : une pour les clés de partition uniquement, et une pour les paires de clés de partition et de tri.
- Facturation en mode clés limitées : dans ce mode, seules les demandes limitées génèrent des événements facturables. Les événements ne sont générés que lorsque les demandes entraînent des erreurs `ProvisionedThroughputExceededException`, `ThrottlingException` ou `RequestLimitExceeded`. Si une table ou un index secondaire global inclut des clés de tri, chaque élément limité représente deux événements (suivi des clés de partition et suivi des clés de partition et de tri).

Exemples de facturation

Par exemple, supposons que votre application effectue les opérations DynamoDB suivantes : un `GetItem`, un `PutItem` et un `BatchWriteItem` qui insère cinq éléments. Supposons également que l'opération `PutItem` soit limitée, mais que toutes les autres opérations réussissent.

- Mode clés consultées et limitées
  - Si votre table ou votre index secondaire global ne comporte qu'une clé de partition, il en résulte 7 événements (1 pour le`GetItem`, 1 pour le `PutItem` et 5 pour le`BatchWriteItem`).
  - Si votre table ou votre index secondaire global possède des clés de partition et des clés de tri, il en résulte 14 événements (2 pour le`GetItem`, 2 pour le `PutItem` et 10 pour le`BatchWriteItem`).
- Mode clés limitées
  - Si votre table ou votre index secondaire global ne contient qu'une clé de partition, il en résulte un événement (uniquement pour le throttled`PutItem`).
  - Si votre table ou votre index secondaire global possède des clés de partition et des clés de tri, cela entraîne 2 événements (2 pour le throttled`PutItem`).

  Les opérations réussies `GetItem` et `BatchWriteItem` ne génèrent aucun événement en mode clés limitées.

Facteurs de facturation courants

Une opération `Query` génère toujours 1 événement, quel que soit le mode ou le nombre d'éléments retournés.

Contrairement aux autres fonctionnalités de DynamoDB CloudWatch , la facturation de Contributor Insights for DynamoDB ne varie pas en fonction des éléments suivants :

- Le mode de capacité (allouée par rapport à la demande)
- Selon que vous exécutiez des requêtes en lecture ou en écriture
- La taille (Ko) des éléments lus ou écrits

Mise en route avec CloudWatch Contributor Insights for DynamoDB

Cette section décrit comment activer et utiliser Amazon CloudWatch Contributor Insights dans différents modes afin de répondre à vos besoins de surveillance à l'aide de la console Amazon DynamoDB ou de l'AWS Command Line Interface (AWS CLI).

Dans les exemples suivants, vous utilisez la table DynamoDB qui est définie dans le didacticiel Mise en route avec DynamoDB.

Rubriques

- Choix d'un mode Contributor Insights
- Utilisation de Contributor Insights (console)
- Utilisation de Contributor Insights (AWS CLI)

Choix d'un mode Contributor Insights

Avant d'activer Contributor Insights, vous devez connaître les deux modes disponibles. Passez en revue la comparaison des modes pour sélectionner l'option qui correspond le mieux à vos besoins spécifiques.

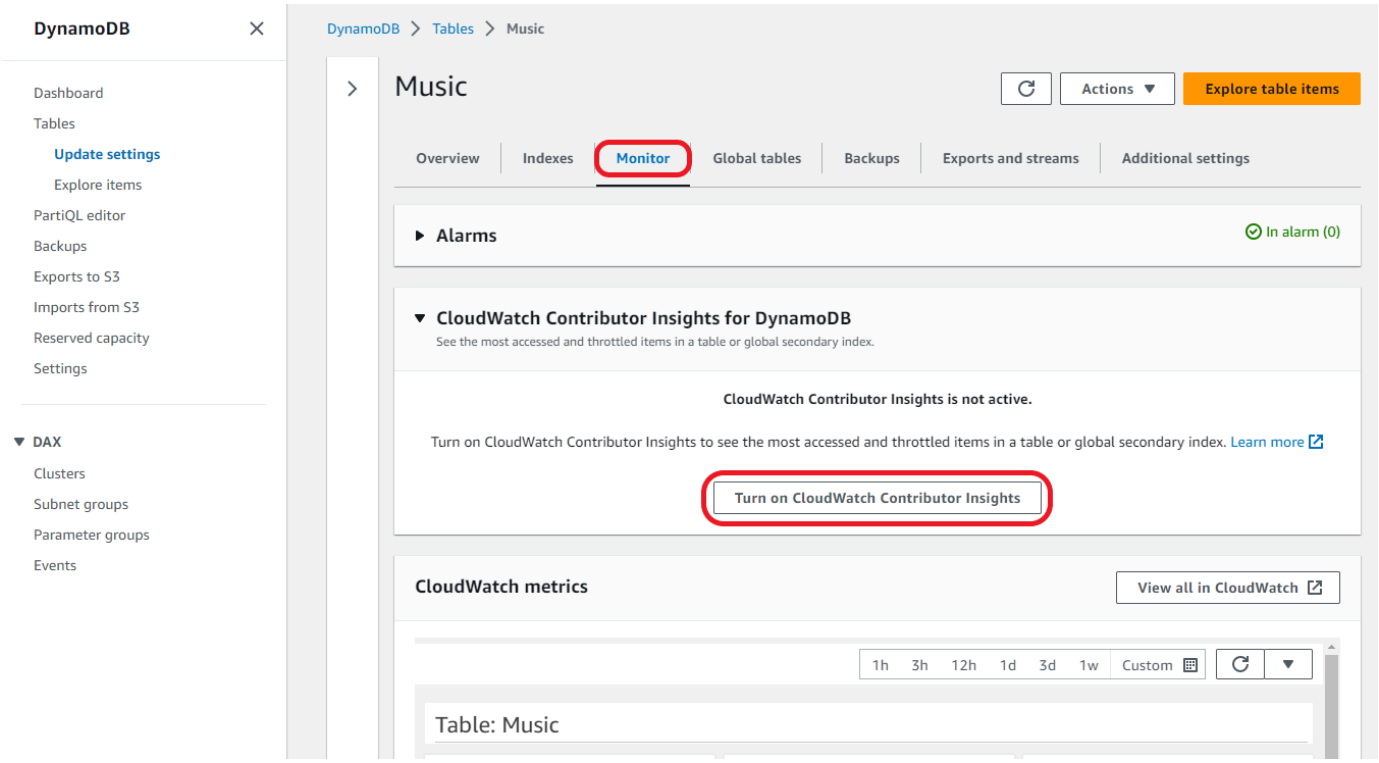
Aspect	Mode clés consultées et limitées	Mode clés limitées
Contrôles	Toutes les demandes (réussies et limitées)	Demandes en mode limité uniquement
Graphiques	Éléments ayant le plus grand nombre d'accès + Éléments les plus limités	Éléments les plus limités uniquement
Idéal pour	Analyse et optimisation ciblées	Surveillance de la limitation
À utiliser lorsque	Vous avez besoin d'une visibilité complète sur les modèles d'accès. Vous effectuez une analyse ou un débogage à court terme.	Votre principale préoccupation est d'identifier et de résoudre les problèmes de limitation. Vous souhaitez que Contributor Insights reste activé en permanence pour les alertes de limitation en temps réel.

Utilisation de Contributor Insights (console)

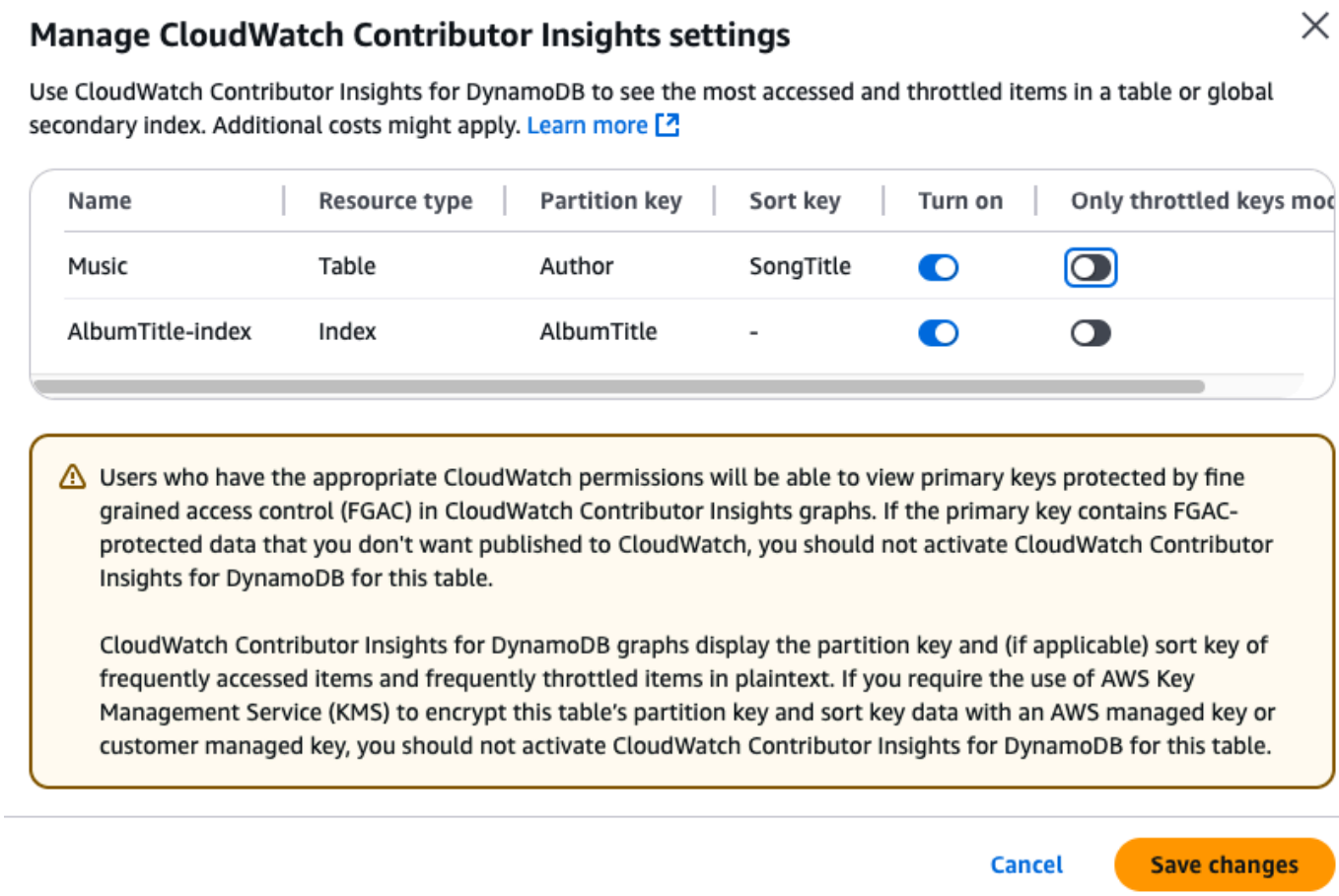
La console fournit un moyen intuitif d'activer Contributor Insights et de sélectionner le mode approprié à vos besoins de surveillance.

Pour utiliser Contributor Insights dans la console

1. Connectez-vous à l'AWS Management Console et ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez la table `Music`.
4. Choisissez l'onglet Surveiller.
5. Choisissez Activer CloudWatch Contributor Insights.



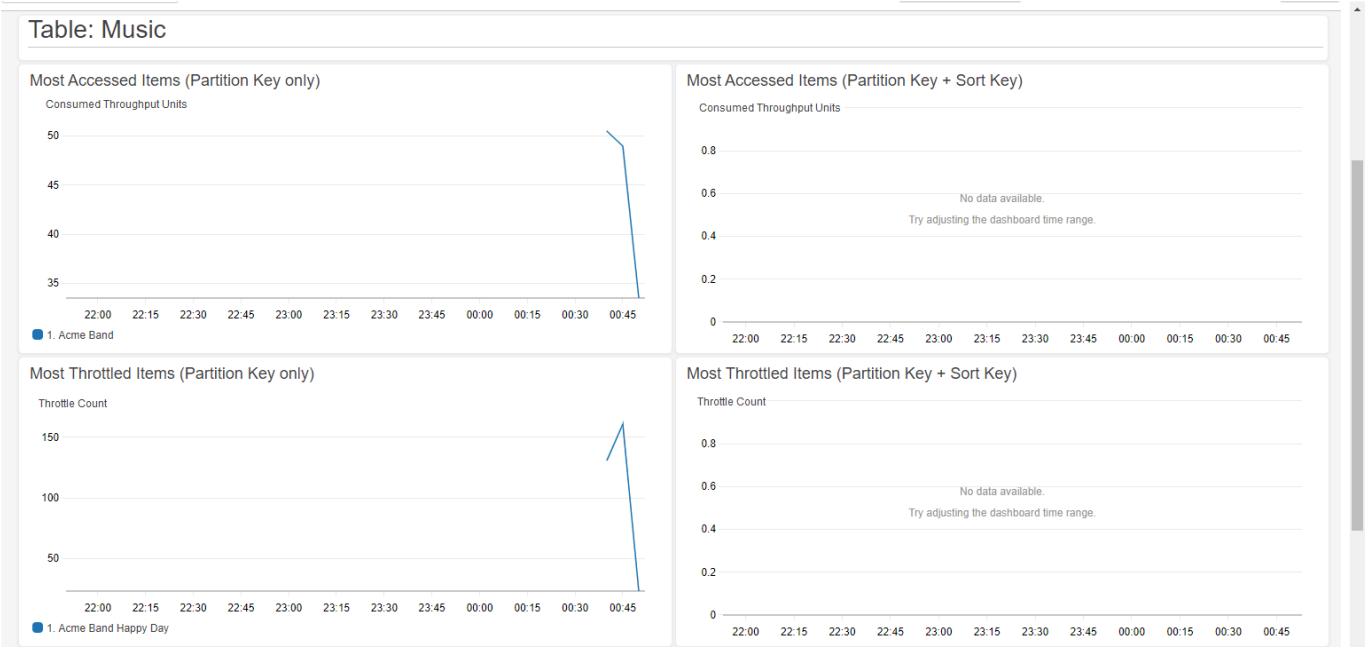
6. Dans la boîte de dialogue Gérer les paramètres de CloudWatch Contributor Insights, activez l'option Activer à la fois pour la table de base `Music` et pour l'index secondaire global `AlbumTitle-index`.
7. Laissez le commutateur Mode clés limitées uniquement en position désactivée pour les deux, puis sélectionnez Enregistrer les modifications.



Cela active le mode clés consultées et limitées par défaut pour la table et le GSI, qui permet de surveiller les éléments consultés et limités. Placer le commutateur du Mode clés limitées uniquement en position activée permettrait d'activer le mode clés limitées.

Si l'opération échoue, consultez DescribeContributorInsights FailureException dans la Référence d'API Amazon DynamoDB afin de connaître les raisons possibles.

8. Les graphiques CloudWatch Contributor Insights sont désormais visibles sur l'onglet Contrôle pour la table `Music`. Comme vous avez activé le mode clés consultées et limitées, vous voyez à la fois les graphiques des éléments consultés et des éléments limités.



Basculement entre les modes

Vous pouvez passer d'un mode à l'autre à tout moment sans désactiver Contributor Insights.

Pour basculer entre les modes de Contributor Insights

1. Dans l'onglet Contrôle de votre table, choisissez Gérer CloudWatch Contributor Insights.
2. Dans la boîte de dialogue Gérer les paramètres de Contributor Insights, pour chaque table de base ou GSI :
   - Activez ou désactivez le Mode clés limitées uniquement pour activer le mode clés limitées ou revenez au mode clés consultées et limitées par défaut.
   - Basculez l'option Activer sur désactivé pour désactiver CloudWatch Contributor Insights pour une table ou un GSI.
3. Sélectionnez Enregistrer les modifications.

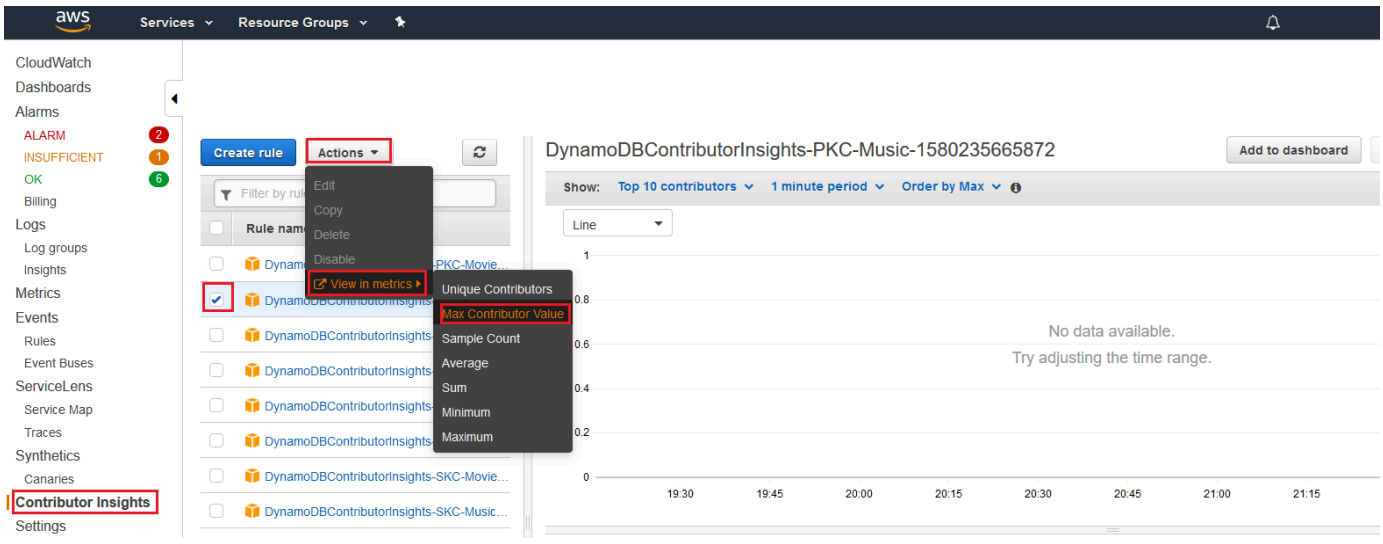
   Une fois que vous avez terminé, les graphiques reflètent le nouveau mode.

Création d'alarmes CloudWatch

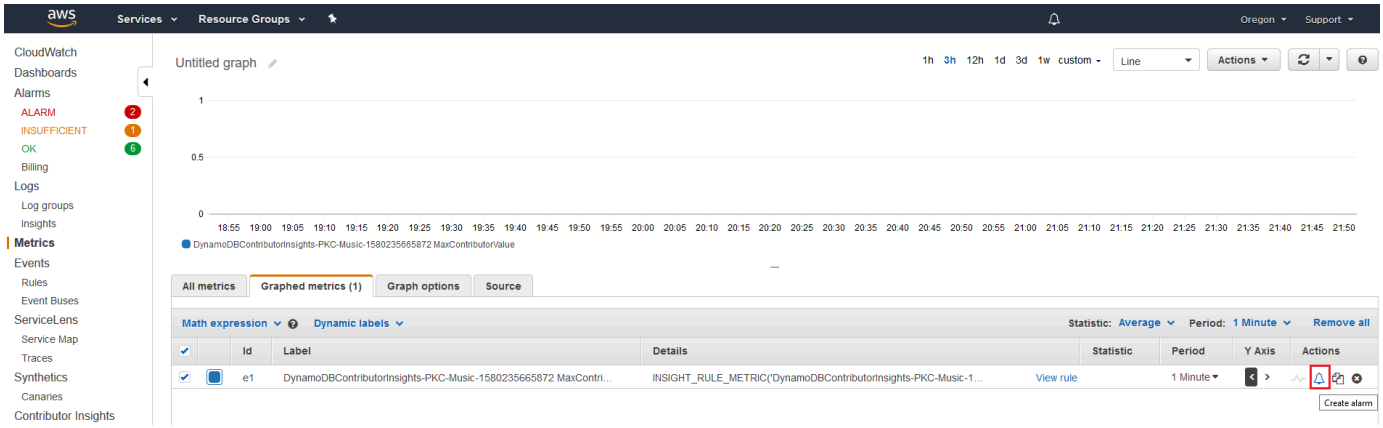
Suivez ces étapes pour créer une alarme CloudWatch et être averti lorsqu'une clé de partition consomme plus de 50 000 ConsumedThroughputUnits ou rencontre une limitation.

1. Connectez-vous à AWS Management Console et ouvrez la console CloudWatch à l'adresse https://console.aws.amazon.com/cloudwatch/
2. Dans le panneau de navigation situé sur le côté gauche de la console, choisissez Contributor Insights.
3. Choisissez la règle appropriée en fonction de votre mode et de ce que vous souhaitez surveiller :
   - Pour la surveillance des éléments consultés (mode clés consultées et limitées uniquement) : choisissez DynamoDBContributorInsights-PKC-Music.
   - Pour la surveillance des éléments limités (les deux modes) : choisissez DynamoDBContributorInsights-PKT-Music.
4. Choisissez la liste déroulante Actions.
5. Choisissez View in metrics (Afficher dans les métriques).
6. Choisissez Max Contributor Value (Valeur maximale du contributeur).

> **Note**
>
> Seules `Max Contributor Value` et `Maximum` renvoient des statistiques utiles. Les autres statistiques de cette liste ne renvoient pas de valeurs intéressantes.

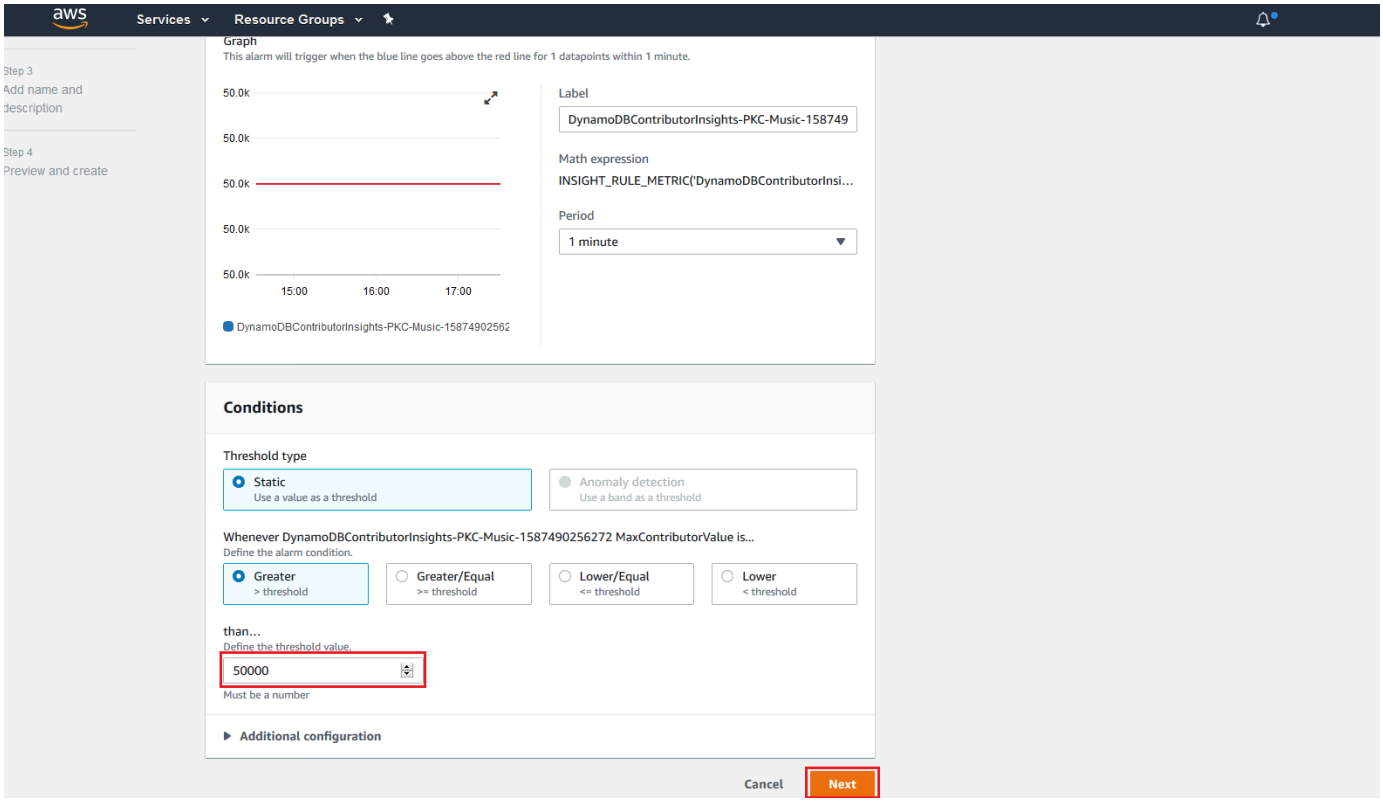


7. Dans la colonne Actions choisissez Create Alarm (Créer une alarme).



8. Entrez une valeur de seuil appropriée et choisissez Suivant :

   - Pour les éléments consultés (règles PKC) : entrez 50 000 pour `ConsumedThroughputUnits`.
   - Pour les éléments limités (règles PKT) : entrez 1 pour `ThrottleCount` pour être alerté en cas de limitation.



9. Consultez Utilisation des alarmes Amazon CloudWatch pour plus d'informations sur la manière de configurer la notification d'alarme.

Utilisation de Contributor Insights (AWS CLI)

L'AWS CLI fournit un accès programmatique à Contributor Insights avec un support complet pour les deux modes. Vous pouvez spécifier le mode lorsque vous activez Contributor Insights ou changer de mode ultérieurement.

Opérations de base avec le mode par défaut

Pour utiliser Contributor Insights avec les paramètres par défaut

1. Activez CloudWatch Contributor Insights for DynamoDB sur la table de base `Music` avec le mode clés consultées et limitées. Comme `ACCESSED_AND_THROTTLED_KEYS` est le mode par défaut, vous pouvez omettre le paramètre `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --contributor-insights-action=ENABLE
   ```

2. Activez Contributor Insights pour DynamoDB sur l'index secondaire global `AlbumTitle-index`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --index-name AlbumTitle-index \
                       --contributor-insights-action=ENABLE
   ```

3. Obtenez le statut et les règles pour la table `Music` et tous ses index.

   ```
   aws dynamodb describe-contributor-insights
                       --table-name Music
   ```

   La réponse inclura le champ `ContributorInsightsMode` indiquant `ACCESSED_AND_THROTTLED_KEYS`.

4. Répertoriez l'état de la table `Music` ainsi que de tous ses index.

   ```
   aws dynamodb list-contributor-insights --table-name Music
   ```

Activation du mode clés limitées

Pour activer Contributor Insights en mode clés limitées

1. Activez CloudWatch Contributor Insights for DynamoDB sur la table de base `Music` avec le mode clés limitées.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. Activez Contributor Insights en mode clés limitées pour l'index secondaire global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Vérifiez le mode en décrivant la configuration de Contributor Insights.

```
aws dynamodb describe-contributor-insights --table-name Music
```

   La réponse affichera `ContributorInsightsMode` comme `THROTTLED_KEYS` et indiquera moins de règles que le mode par défaut.

Basculement entre les modes

Pour basculer entre les modes de Contributor Insights

1. Passez du mode clés limitées au mode clés consultées et limitées.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. Passez du mode clés consultées et limitées au mode clés limitées.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Vérifiez l'état pendant la transition.

```
aws dynamodb describe-contributor-insights --table-name Music
```

Pendant le changement de mode, l'état `ContributorInsightsStatus` indiquera `ENABLING`. Une fois l'opération terminée, il apparaîtra comme `ENABLED` avec le nouveau mode.

Gestion de Contributor Insights

Pour gérer les paramètres de Contributor Insights

1. Désactivez CloudWatch Contributor Insights pour DynamoDB sur l'index secondaire global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. Répertoriez toutes les configurations de Contributor Insights de votre compte.

```
aws dynamodb list-contributor-insights
```

Cela affiche toutes les tables et tous les index pour lesquels Contributor Insights est activé, ainsi que leurs modes.

3. Obtenez des informations détaillées sur une configuration spécifique.

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

Exemple de réponses

Voici des exemples de réponses illustrant les différences entre les modes :

Réponse en mode clés consultées et limitées

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKC-Music-1234567890123",
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKC-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "ACCESSED_AND_THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:30:00.000Z"
}
```

Réponse en mode clés limitées

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

Notez que le mode clés limitées comporte moins de règles (uniquement PKT et SKT), ce qui correspond à une surveillance plus ciblée.

Utilisation d'IAM avec les informations des CloudWatch contributeurs pour DynamoDB

La première fois que vous activez Amazon CloudWatch Contributor Insights pour Amazon DynamoDB, DynamoDB Gestion des identités et des accès AWS crée automatiquement un rôle lié à un service (IAM) pour vous. Ce rôle permet à DynamoDB de CloudWatch gérer les règles de Contributor Insights en votre nom. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Ne supprimez pas ce rôle lié à un service. Si vous le supprimez, toutes vos règles gérées ne seront plus nettoyées lorsque vous supprimerez votre table ou votre index secondaire global.

Pour en savoir plus sur l'utilisation des rôles liés à un service, consultez Utilisation des rôles liés à un service dans le Guide de l'utilisateur IAM.

Les autorisations suivantes sont requises :

- Pour activer ou désactiver CloudWatch Contributor Insights pour DynamoDB, vous devez `dynamodb:UpdateContributorInsights` disposer d'une autorisation sur la table ou l'index.
- Pour consulter les graphiques CloudWatch Contributor Insights for DynamoDB, vous devez disposer d'une autorisation. `cloudwatch:GetInsightRuleReport`
- Pour décrire CloudWatch Contributor Insights for DynamoDB pour une table ou un index DynamoDB donné, vous devez disposer d'une autorisation. `dynamodb:DescribeContributorInsights`
- Pour répertorier les statuts DynamoDB de CloudWatch Contributor Insights pour chaque table et index secondaire global, vous devez disposer d'une autorisation. `dynamodb:ListContributorInsights`

Exemple : activer ou désactiver les informations sur les CloudWatch contributeurs pour DynamoDB

La politique IAM suivante accorde des autorisations pour activer ou désactiver CloudWatch Contributor Insights pour DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

Pour les tables chiffrées par clé KMS, l'utilisateur doit disposer des autorisations de kms: déchiffrement afin de mettre à jour Contributor Insights.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
```

```
        }
    ]
}
```

Exemple : rapport sur les règles de récupération des informations sur les CloudWatch contributeurs

La politique IAM suivante accorde les autorisations nécessaires pour récupérer le rapport des règles de CloudWatch Contributor Insights.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Exemple : appliquer de manière sélective les informations des CloudWatch contributeurs pour les autorisations DynamoDB en fonction des ressources

La politique IAM suivante autorise les actions `ListContributorInsights` et `DescribeContributorInsights`, et rejette l'action `UpdateContributorInsights` pour un index secondaire global spécifique.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
}
```

Utilisation de rôles liés à un service pour CloudWatch Contributor Insights pour DynamoDB

CloudWatch Contributor Insights for DynamoDB Gestion des identités et des accès AWS utilise des rôles liés à un service (IAM). Un rôle lié à un service est un type unique de rôle IAM directement lié à CloudWatch Contributor Insights for DynamoDB. Les rôles liés à un service sont prédéfinis par CloudWatch Contributor Insights for DynamoDB et incluent toutes les autorisations dont le service a besoin pour appeler d'autres services en votre nom.AWS

Un rôle lié à un service facilite la configuration de CloudWatch Contributor Insights pour DynamoDB, car il n'est pas nécessaire d'ajouter manuellement les autorisations nécessaires. CloudWatch Contributor Insights for DynamoDB définit les autorisations associées à ses rôles liés aux services et, sauf indication contraire, seul CloudWatch Contributor Insights for DynamoDB peut assumer ses rôles. Les autorisations définies comprennent la politique d'approbation et la politique d'autorisation. De plus, cette politique d'autorisation ne peut pas être attachée à une autre entité IAM.

Pour plus d'informations sur les autres services qui prennent en charge les rôles liés à un service, veuillez consulter Services AWS qui fonctionnent avec IAM et recherchez les services où Oui figure dans la colonne Rôle lié à un service. Choisissez un Oui ayant un lien permettant de consulter les détails du rôle pour ce service.

Autorisations de rôle liées à un service pour CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB utilise le rôle lié à un service nommé. AWSServiceRoleForDynamoDBCloudWatchContributorInsights L'objectif du rôle lié à un service est de permettre à Amazon DynamoDB de gérer CloudWatch les règles Amazon Contributor Insights créées pour les tables DynamoDB et les index secondaires globaux, en votre nom.

Le rôle lié à un service `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` approuve les services suivants pour endosser le rôle :

- `contributorinsights.dynamodb.amazonaws.com`

La politique d'autorisation des rôles permet à CloudWatch Contributor Insights for DynamoDB d'effectuer les actions suivantes sur les ressources spécifiées :

- Action : `Create and manage Insight Rules` sur `DynamoDBContributorInsights`

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Pour en savoir plus, consultez Service-Linked Role Permissions (autorisations du rôle lié à un service) dans le Guide de l’utilisateur IAM.

Création d'un rôle lié à un service pour CloudWatch Contributor Insights for DynamoDB

Vous n’avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous activez Contributor Insights dans AWS Management Console, dans ou dans l'AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB crée pour vous le rôle lié au service.

Si vous supprimez ce rôle lié à un service et que vous avez ensuite besoin de le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous activez Contributor Insights, CloudWatch Contributor Insights for DynamoDB crée à nouveau le rôle lié au service pour vous.

Modification d'un rôle lié à un service pour CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB ne vous permet pas de modifier `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` le rôle lié à un service. Après avoir créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence à ce rôle. Néanmoins, vous pouvez modifier la description du rôle à

l’aide d’IAM. Pour plus d’informations, consultez Modification d’un rôle lié à un service dans le Guide de l’utilisateur IAM.

Suppression d'un rôle lié à un service pour CloudWatch Contributor Insights for DynamoDB

Vous n’avez pas besoin de supprimer manuellement le rôle `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Lorsque vous désactivez Contributor Insights dans AWS Management Console, dans ou dans l'AWS API, CloudWatch Contributor Insights for DynamoDB nettoie les ressources.AWS CLI

Vous pouvez également utiliser la console IAM AWS CLI ou l'AWS API pour supprimer manuellement le rôle lié au service. Pour ce faire, vous devez commencer par nettoyer les ressources de votre rôle lié à un service. Vous pouvez ensuite supprimer ce rôle manuellement.

> **Note**
>
> Si le service CloudWatch Contributor Insights for DynamoDB utilise le rôle lorsque vous essayez de supprimer les ressources, la suppression risque d'échouer. Si cela se produit, patientez quelques minutes et réessayez.

Pour supprimer manuellement le rôle lié au service à l’aide d’IAM

Utilisez la console IAM AWS CLI, le ou l'AWS API pour supprimer le rôle lié au `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` service. Pour plus d’informations, consultez la section Suppression d’un rôle lié à un service dans le Guide de l’utilisateur IAM.

Bonnes pratiques de conception et d'architecture avec DynamoDB

Utilisez cette section comme référence pour trouver rapidement les recommandations relatives à l'optimisation des performances et à la réduction des coûts de débit lors de l'utilisation de DynamoDB.

Rubriques

- Conception NoSQL pour DynamoDB
- Utilisation du cadre DynamoDB Well-Architected pour optimiser votre charge de travail DynamoDB
- Bonnes pratiques pour la conception et l'utilisation performantes de clés de partition dans DynamoDB
- Bonnes pratiques concernant l'utilisation de clés de tri pour organiser les données dans DynamoDB
- Bonnes pratiques d'utilisation d'index secondaires dans DynamoDB
- Bonnes pratiques de stockage d'éléments volumineux et d'attributs dans DynamoDB
- Bonnes pratiques de gestion des données de séries temporelles dans DynamoDB
- Meilleures pratiques pour gérer les many-to-many relations dans les tables DynamoDB
- Bonnes pratiques pour l'interrogation et l'analyse de données dans DynamoDB
- Bonnes pratiques relatives à la conception d'une table DynamoDB
- Utilisation de tables globales DynamoDB
- Bonnes pratiques pour gérer le plan de contrôle dans DynamoDB
- Bonnes pratiques liées à l'utilisation d'opérations de données groupées dans DynamoDB
- Bonnes pratiques relatives à la mise en œuvre du contrôle de version dans DynamoDB
- Bonnes pratiques pour comprendre vos rapports AWS de facturation et d'utilisation dans DynamoDB
- Migration d'une table DynamoDB d'un compte à un autre
- Recommandations pour intégrer DAX aux applications DynamoDB

- Considérations relatives à l'utilisation AWS PrivateLink d'Amazon DynamoDB

Conception NoSQL pour DynamoDB

Les systèmes de base de données NoSQL comme Amazon DynamoDB utilisent d'autres modèles pour la gestion des données, tels que les paires clé-valeur ou le stockage de documents. Il est important de comprendre les principales différences et les approches de conception spécifiques lorsque vous passez d'un système de gestion de base de données relationnelle (SGBDR) à un système de base de données NoSQL tel que DynamoDB.

Rubriques

- Différences entre le système de base de données relationnelle et le système NoSQL
- Deux concepts essentiels de la conception NoSQL
- Approche de la conception NoSQL
- NoSQL Workbench pour DynamoDB

Différences entre le système de base de données relationnelle et le système NoSQL

Les systèmes de gestion de bases de données relationnelles (SGBDR) et les bases de données NoSQL ont chacun leurs forces et leurs faiblesses :

- Dans les SGBDR, les données peuvent être interrogées avec souplesse, mais les requêtes sont relativement coûteuses et ne sont pas bien mises à l'échelle dans les situations de trafic intense (consultez Premiers pas pour la modélisation des données relationnelles dans DynamoDB).
- Dans une base de données NoSQL comme DynamoDB, les données peuvent être interrogées efficacement d'un nombre limité de façons, en dehors desquelles les requêtes peuvent être coûteuses et lentes.

Ces différences entraînent une conception des bases de données très différente d'un système à l'autre :

- Dans un SGBDR, la conception se concentre avant tout sur la flexibilité, sans se préoccuper des détails de l'implémentation ni des performances. En général, l'optimisation des requêtes n'affecte pas la conception du schéma, mais la normalisation est très importante.

- Dans DynamoDB, votre schéma est conçu spécifiquement pour que les requêtes les plus courantes et les plus importantes soient aussi rapides et économiques que possible. Les structures de vos données sont adaptées aux exigences spécifiques de vos cas d'utilisation.

Deux concepts essentiels de la conception NoSQL

Pour concevoir un système NoSQL, il faut un autre état d'esprit que pour un SGBDR. Pour un SGBDR, vous pouvez créer un modèle de données normalisé sans réfléchir aux modèles d'accès. Vous pouvez l'étendre ultérieurement, pour répondre à de nouvelles questions et de nouveaux besoins d'interrogation. Vous pouvez organiser chaque type de données dans sa propre table.

En quoi la conception d'un système NoSQL est différente

- En revanche, vous ne devez pas commencer à concevoir votre schéma pour DynamoDB tant que vous ne savez pas à quelle problématique celui-ci doit répondre. Il est essentiel d'identifier au préalable les problèmes métier et les cas d'utilisation de l'application.
- Vous devez gérer le moins de tables possible dans une application DynamoDB. Le fait d'avoir moins de tables rend les choses plus évolutives, nécessite moins de gestion des autorisations et réduit les frais généraux pour votre application DynamoDB. Cela peut également contribuer à maintenir des coûts de sauvegarde globalement plus faibles.

Approche de la conception NoSQL

L'identification des modèles de requêtes spécifiques que le système doit satisfaire constitue la première étape de la conception de votre application DynamoDB.

Il est notamment important de comprendre trois propriétés fondamentales des modèles d'accès de votre application avant de commencer :

- Taille des données : connaître la quantité de données qui sera stockée et demandée simultanément aide à déterminer le moyen le plus efficace pour partitionner les données.
- Forme des données : au lieu de remodéliser les données lors du traitement d'une requête (comme c'est le cas dans un SGBDR), une base de données NoSQL organise les données de manière à ce que leur forme dans la base de données corresponde aux requêtes. C'est un élément clé pour augmenter la vitesse et la scalabilité.
- Vitesse des données : DynamoDB effectue une mise à l'échelle en augmentant le nombre de partitions physiques disponibles pour traiter les requêtes, et en répartissant de manière efficace

les données sur ces partitions. Le fait de connaître à l'avance les pics de charge des requêtes peut aider à déterminer comment partitionner les données de manière à utiliser au mieux I/O la capacité.

Après avoir identifié les besoins de requête spécifiques, vous pouvez organiser les données en fonction des principes généraux qui déterminent la performance :

- Rassemblez les données connexes. Des recherches ont montré que le principe de « localité de référence », qui consiste à regrouper les données associées en un seul endroit, est un facteur clé pour améliorer les performances et les temps de réponse dans les systèmes NoSQL. Il s'est également avéré important pour optimiser les tables de routage il y a de nombreuses années.

  En règle générale, vous devez gérer le moins de tables possible dans une application DynamoDB.

  Les cas où des données de série chronologique volumineuses sont impliquées ou dans lesquels les ensembles de données ont des modèles d'accès très différents sont des exemples d'exceptions. Une table unique avec des index inversés peut généralement permettre à des requêtes simples de créer et récupérer les structures de données hiérarchiques complexes dont votre application a besoin.
- Utilisez l'ordre de tri. Les éléments connexes peuvent être regroupés et interrogés de manière efficace si la conception des clés permet de les trier ensemble. Il s'agit d'une stratégie de conception NoSQL importante.
- Répartissez les requêtes. Il est également important qu'un volume élevé de requêtes ne soit pas focalisé sur une partie de la base de données, où elles peuvent dépasser la I/O capacité. Au lieu de cela, vous devez concevoir les clés de données de manière à ce que le trafic soit, autant que possible, répartit de manière uniforme sur les partitions, évitant ainsi les zones sensibles.
- Utilisez des index secondaires globaux. En créant des index secondaires globaux spécifiques, vous pouvez accepter différentes requêtes que votre table principale peut prendre en charge, et qui restent rapides et relativement économiques.

Ces principes généraux se traduisent en modèles de conception courants disponibles pour modéliser les données de manière efficace dans DynamoDB.

NoSQL Workbench pour DynamoDB

NoSQL Workbench pour DynamoDB est une application côté client multiplateforme dotée d'une interface utilisateur graphique dont vous pouvez vous servir pour développer et exploiter des bases de données modernes. Il est disponible pour Windows, macOS et Linux. NoSQL Workbench est

un outil de développement visuel qui propose des fonctionnalités de modélisation de données, de visualisation de données, de génération de données d'exemple et de développement de requêtes, afin de vous aider à concevoir, créer, interroger et gérer des tables DynamoDB. NoSQL Workbench pour DynamoDB vous permet de créer des modèles de données ou de concevoir des modèles à partir de modèles de données existants, qui pourront satisfaire les modèles d'accès aux données de votre application. À la fin du processus, vous pouvez également importer et exporter le modèle de données conçu. Pour de plus amples informations, veuillez consulter Création de modèles de données avec NoSQL Workbench.

Utilisation du cadre DynamoDB Well-Architected pour optimiser votre charge de travail DynamoDB

Cette section décrit le cadre DynamoDB Well-Architected, un ensemble de principes de conception et de conseils pour concevoir des charges de travail DynamoDB bien architecturées.

Optimisation des coûts sur les tables DynamoDB

Cette section décrit les meilleures pratiques permettant d'optimiser les coûts pour vos tables DynamoDB existantes. Vous devez examiner les stratégies suivantes pour déterminer la stratégie d'optimisation des coûts qui répond le mieux à vos besoins et les aborder de manière itérative. Chaque stratégie fournira un aperçu de ce qui pourrait avoir un impact sur vos coûts, des signes à rechercher et des conseils prescriptifs sur la manière de les réduire.

Rubriques

- Évaluer les coûts au niveau de la table
- Évaluation du mode de capacité de votre table DynamoDB
- Évaluation des paramètres d'autoscaling de votre table DynamoDB
- Évaluation de la sélection de votre classe de table DynamoDB
- Identification de vos ressources inutilisées dans DynamoDB
- Évaluer les modèles d'utilisation de votre table DynamoDB
- Évaluation de l'utilisation de vos flux DynamoDB
- Évaluation de votre capacité provisionnée pour un dimensionnement approprié dans votre table DynamoDB

Évaluer les coûts au niveau de la table

L'outil Cost Explorer qui se trouve dans le AWS Management Console permet de visualiser les coûts ventilés par type, tels que les frais de lecture, d'écriture, de stockage et de sauvegarde. Vous pouvez également voir ces coûts résumés par période, par mois ou par jour.

Les administrateurs doivent notamment faire face au problème de révision nécessaire des coûts d'une seule table en particulier. Certaines de ces données sont disponibles via la console DynamoDB ou via des appels à l'API `DescribeTable`, mais Cost Explorer ne vous permet pas, par défaut, de filtrer ou de regrouper en fonction des coûts associés à une table spécifique. Cette section explique comment utiliser le balisage pour effectuer une analyse des coûts de chaque table dans Cost Explorer.

Rubriques

- Comment consulter les coûts d'une seule table DynamoDB
- Vue par défaut de Cost Explorer
- Comment utiliser et appliquer des balises de table dans Cost Explorer

Comment consulter les coûts d'une seule table DynamoDB

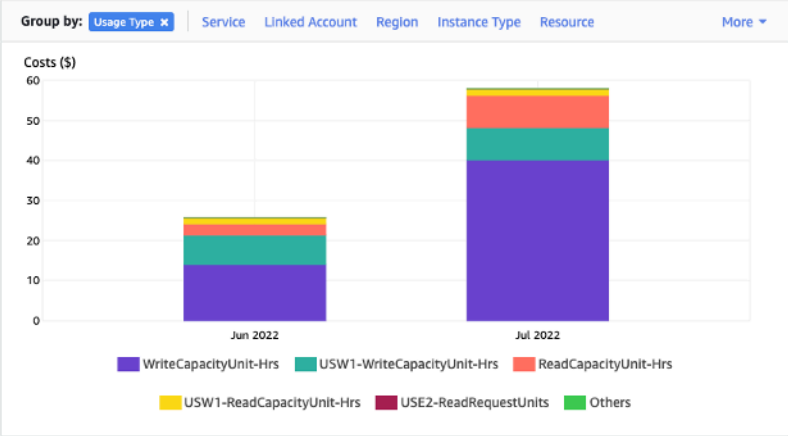
Amazon AWS Management Console DynamoDB et l'API vous fourniront des informations sur une seule table, notamment `DescribeTable` le schéma de clé primaire, tous les index de la table, ainsi que la taille et le nombre d'éléments de la table et de tous les index. La taille de la table et celle des index peuvent être utilisées pour calculer le coût de stockage mensuel de votre table. Par exemple, 0,25 USD par Go dans la région us-east-1.

Si la table est en mode de capacité provisionnée, les paramètres RCU et WCU actuels sont également renvoyés. Ils peuvent être utilisés pour calculer les coûts de lecture et d'écriture actuels de la table, mais ces coûts peuvent changer, en particulier si la table a été configurée avec Auto Scaling.

> **Note**
>
> Si la table est en mode de capacité à la demande, `DescribeTable` n'aidera pas à estimer les coûts de débit, car ceux-ci sont facturés en fonction de l'utilisation réelle et non provisionnée au cours d'une période donnée.

Vue par défaut de Cost Explorer

La vue par défaut de Cost Explorer fournit des graphiques indiquant le coût des ressources consommées, telles que le débit et le stockage. Vous pouvez choisir de regrouper les coûts par période, par exemple les totaux par mois ou par jour. Les coûts de stockage, de lecture, d'écriture et d'autres fonctionnalités peuvent également être ventilés et comparés.



Comment utiliser et appliquer des balises de table dans Cost Explorer

Par défaut, Cost Explorer ne fournit pas de résumé des coûts pour une table spécifique, car il combinera les coûts de plusieurs tables pour former un total. Vous pouvez toutefois utiliser le balisage des ressources AWS pour identifier chaque table à l'aide d'une balise de métadonnées. Les balises sont des paires clé-valeur que vous pouvez utiliser à des fins diverses, par exemple pour identifier toutes les ressources appartenant à un projet ou à un département. Pour cet exemple, nous supposerons que vous avez une table nommée MyTable.

1. Définissez une balise avec la clé table_name et la valeur de. MyTable
2. Activez la balise dans Cost Explorer, puis filtrez sur la valeur de la balise pour obtenir une meilleure visibilité sur les coûts de chaque table.

> **Note**
>
> Cela peut prendre un ou deux jours pour que la balise commence à apparaître dans Cost Explorer

Vous pouvez définir vous-même les balises de métadonnées dans la console ou via une automatisation telle que la AWS CLI ou le AWS SDK. Envisagez d'exiger la définition d'une balise

table_name dans le cadre du nouveau processus de création de tables de votre organisation. Pour les tables existantes, un utilitaire Python permet de rechercher et d'appliquer ces balises à toutes les tables existantes d'une certaine région de votre compte. Voir Eponymous Table Tagger sur GitHub pour plus de détails.

Évaluation du mode de capacité de votre table DynamoDB

Cette section explique comment sélectionner le mode de capacité approprié pour votre table DynamoDB. Chaque mode est réglé de façon à répondre aux besoins d'une charge de travail différente en termes de réactivité face à l'évolution du débit, ainsi que de facturation de cette utilisation. Vous devez tenir compte de ces facteurs lorsque vous prenez votre décision.

Rubriques

- Quels sont les modes de capacité de table disponibles ?
- Quand sélectionner le mode de capacité à la demande ?
- Quand sélectionner le mode de capacité provisionnée ?
- Autres facteurs à prendre en compte lors du choix d'un mode de capacité de table

Quels sont les modes de capacité de table disponibles ?

Lorsque vous créez une table DynamoDB, vous devez sélectionner le mode de capacité à la demande ou de capacité provisionnée.

Vous pouvez faire passer les tables du mode de capacité provisionnée au mode à la demande jusqu'à quatre fois par période de 24 heures. Vous pouvez à tout moment faire passer des tables du mode à la demande au mode de capacité provisionnée.

Mode de capacité à la demande

Le mode de capacité à la demande est conçu pour éliminer la nécessité de planifier ou de provisionner la capacité de votre table DynamoDB. Dans le cadre de ce mode, votre table répondra instantanément aux demandes qui lui sont adressées sans qu'il soit nécessaire d'augmenter ou de réduire les ressources (jusqu'à deux fois le débit de pointe précédent de la table).

DynamoDB à la demande pay-per-request propose des tarifs pour les demandes de lecture et d'écriture afin que vous ne payiez que pour ce que vous utilisez.

Mode de capacité provisionnée

Le mode de capacité provisionnée est un modèle plus traditionnel dans le cadre duquel vous devez définir la capacité disponible de la table pour les demandes, soit directement, soit à l'aide de l'autoscaling. Dans la mesure où une capacité spécifique est mise en service pour la table à tout moment, la facturation est basée sur la capacité totale provisionnée plutôt que sur le nombre de demandes consommées. Le dépassement de la capacité provisionnée peut également entraîner le rejet des demandes par la table et réduire l'expérience des utilisateurs de vos applications.

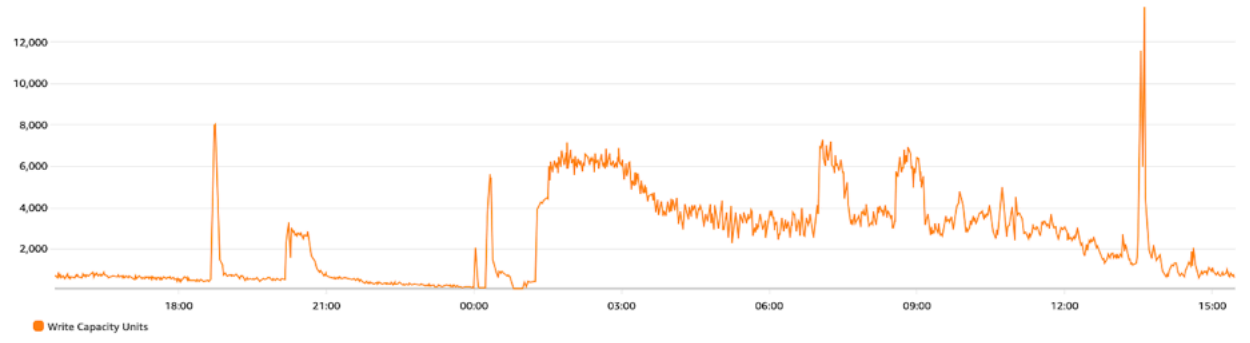
Le mode de capacité provisionnée nécessite une surveillance constante pour trouver un équilibre entre le provisionnement excessif ou insuffisant de la table, afin de limiter les contraintes et de maîtriser les coûts.

Quand sélectionner le mode de capacité à la demande ?

Lors de l'optimisation des coûts, le mode à la demande est le meilleur choix lorsque vous avez une charge de travail similaire à celle des graphiques suivants.

Les facteurs suivants contribuent à ce type de charge de travail :

- Schéma de trafic qui évolue au fil du temps
- Volume variable des demandes (en raison des charges de travail par lots)
- Caractère imprévisible des demandes (entraînant des pics de trafic)
- Tombe à zéro ou en dessous de 30 % du pic à une heure donnée



Pour les charges de travail présentant les facteurs ci-dessus, l'utilisation de l'autoscaling pour maintenir une capacité suffisante sur la table afin de répondre aux pics de trafic risque d'entraîner un provisionnement excessif de la table et un coût supérieur à ce qui est nécessaire, ou un provisionnement insuffisant de la table et une limitation inutile des demandes. Le mode de capacité à la demande est le meilleur choix, car il permet de gérer les fluctuations du trafic sans que vous ayez à prévoir ou à ajuster la capacité.

Grâce au modèle de pay-per-request tarification du mode à la demande, vous n'avez pas à vous soucier de la capacité inutilisée, car vous ne payez que pour le débit que vous utilisez réellement. Vous êtes facturé par demande de lecture ou d'écriture consommée, de sorte que vos coûts reflètent directement votre utilisation réelle. Cela permet d'équilibrer plus facilement les coûts et les performances. Vous pouvez également configurer le débit maximal de lecture ou d'écriture (ou les deux) par seconde pour les tables individuelles à la demande et les index secondaires globaux, afin de contenir les coûts et l'utilisation. Pour plus d'informations, consultez maximum throughput for on-demand tables.

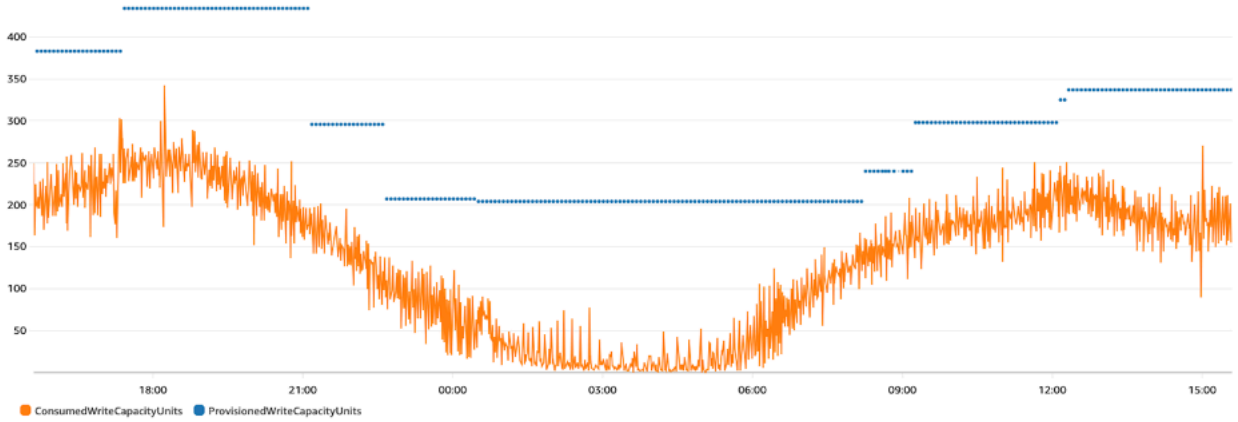
Quand sélectionner le mode de capacité provisionnée ?

Une charge de travail idéale pour le mode de capacité provisionnée présente un modèle d'utilisation plus fiable et prévisible, comme le montre le graphique ci-dessous.

> **Note**
>
> Nous vous recommandons de passer en revue les indicateurs sur une période précise, par exemple 14 jours, avant de prendre des mesures sur votre capacité provisionnée.

Les facteurs suivants contribuent à ce type de charge de travail :

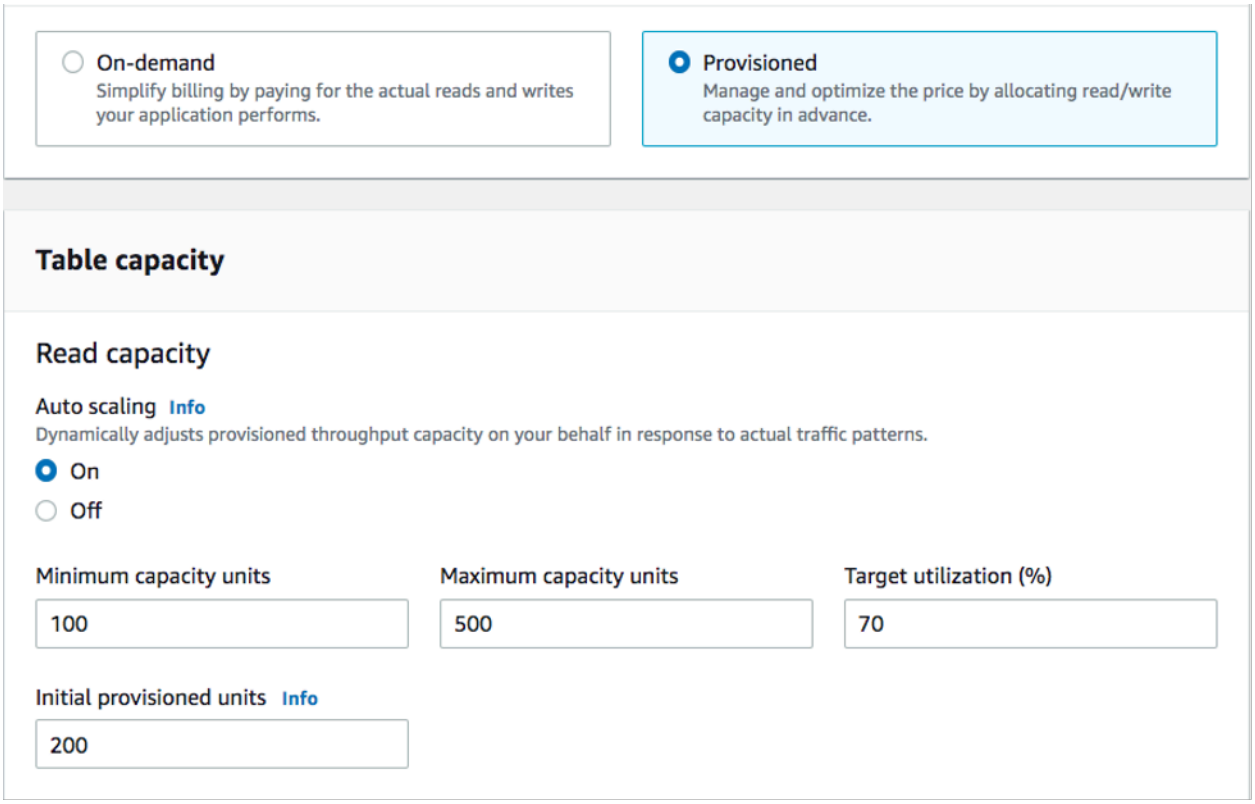
- Trafic régulier, prévisible et cyclique pour une heure ou une journée donnée
- Rafales de trafic limitées à court terme



Étant donné que les volumes de trafic au cours d'une heure ou d'une journée donnée sont plus stables, vous pouvez définir la capacité provisionnée de la table relativement proche de la capacité

réellement consommée de celle-ci. L'optimisation des coûts d'une table à capacité provisionnée consiste en fin de compte à rapprocher la capacité provisionnée (ligne bleue) de la capacité consommée (ligne orange) le plus possible sans augmenter les `ThrottledRequests` sur la table. L'espace entre les deux lignes est à la fois une perte de capacité et une assurance contre une mauvaise expérience utilisateur due aux limitations. Si vous pouvez prévoir les exigences de débit de votre application et que vous préférez la prévisibilité des coûts liée au contrôle de la capacité de lecture et d'écriture, vous pouvez continuer à utiliser des tables provisionnées.

DynamoDB fournit une mise à l'échelle automatique pour les tables à capacité provisionnée, ce qui créera automatiquement un équilibre. Cela vous permet de suivre votre capacité consommée tout au long de la journée et de définir la capacité de la table en fonction de différentes variables. Lorsque vous utilisez l'autoscaling, votre table sera surprovisionnée et vous devrez optimiser le ratio entre le nombre de limitations et le nombre d'unités de capacité surprovisionnées pour répondre aux besoins de votre charge de travail.



Unités de capacité minimale

Vous pouvez définir la capacité minimale d'une table pour réduire les limitations, mais cela ne réduira pas le coût de la table. Si votre table connaît des périodes de faible utilisation suivies d'une soudaine augmentation de l'utilisation, la définition du minimum peut empêcher la mise à l'échelle automatique de définir la capacité de la table sur une valeur trop faible.

Unité de capacité maximale

Vous pouvez définir la capacité maximale d'une table afin de limiter la mise à l'échelle d'une table en utilisant une valeur supérieure à celle prévue. Envisagez d'appliquer une valeur maximale pour les tables de développement ou de test, car des tests de charge à grande échelle ne sont pas souhaitables dans ces conditions. Vous pouvez définir une valeur maximale pour n'importe quelle table, mais veillez à évaluer régulièrement ce paramètre par rapport à la ligne de base de la table lorsque vous l'utilisez en production afin d'éviter toute limitation accidentelle.

Utilisation cible

La définition de l'utilisation cible de la table est le principal moyen d'optimiser les coûts pour une table à capacité provisionnée. La définition d'une valeur de pourcentage inférieure ici augmentera le niveau de provisionnement excessif de la table, ce qui augmentera les coûts, mais réduira le risque de limitation. La définition d'une valeur de pourcentage plus élevée réduira le niveau de provisionnement excessif de la table, mais augmentera le risque de limitation.

Autres facteurs à prendre en compte lors du choix d'un mode de capacité de table

Lorsque vous choisissez entre les deux modes, certains facteurs supplémentaires méritent d'être pris en compte.

Utilisation de la capacité provisionnée

Pour déterminer dans quels cas le mode à la demande coûterait moins cher que la capacité allouée, il est utile d'examiner l'utilisation de votre capacité provisionnée, qui fait référence à l'efficacité avec laquelle les ressources allouées (ou « provisionnées ») sont utilisées. Le mode à la demande coûte moins cher pour les charges de travail dont l'utilisation moyenne de la capacité provisionnée est inférieure à 35 %. Dans de nombreux cas, même pour les charges de travail dont l'utilisation de la capacité provisionnée est supérieure à 35 %, il peut être plus rentable d'utiliser le mode à la demande, en particulier si la charge de travail présente des périodes de faible activité associées à des pics occasionnels.

Capacité réservée

Pour les tables à capacité provisionnée, DynamoDB offre la possibilité d'acheter de la capacité réservée pour votre capacité de lecture et d'écriture (les unités de capacité d'écriture répliquées (rWCU) et les tables Standard-IA ne sont pas éligibles pour l'instant). La capacité réservée offre des remises importantes par rapport à la tarification standard de la capacité provisionnée.

Lorsque vous choisissez entre les deux modes de table, considérez dans quelle mesure cette remise supplémentaire affectera le coût de la table. Dans certains cas, même une charge de travail relativement imprévisible peut être moins coûteuse à exécuter sur une table à capacité provisionnée et provisionnement excessif avec une capacité réservée.

Améliorer la prévisibilité de votre charge de travail

Dans certaines situations, une charge de travail peut sembler à la fois prévisible et imprévisible. Bien que cela puisse être facilement pris en charge par une table à la demande, les coûts seront probablement plus intéressants si les modèles imprévisibles de la charge de travail peuvent être améliorés.

L'une des raisons les plus courantes est l'importation par lots. Ce type de trafic peut souvent dépasser la capacité de base de la table à un point tel qu'une limitation se produirait si elle était exécutée. Pour qu'une charge de travail comme celle-ci soit exécutée sur une table à capacité provisionnée, considérez les options suivantes :

- Si le traitement par lots se produit à des heures planifiées, vous pouvez planifier une augmentation de votre capacité de mise à l'échelle automatique avant son exécution
- Si le traitement par lots se produit de manière aléatoire, pensez à essayer de prolonger sa durée d'exécution au lieu de l'exécuter le plus rapidement possible
- Ajoutez une période d'accélération à l'importation pendant laquelle la vitesse de l'importation est faible au départ, mais augmente lentement en quelques minutes jusqu'à ce que l'autoscaling ait eu l'occasion de commencer à ajuster la capacité de la table

Évaluation des paramètres d'autoscaling de votre table DynamoDB

Cette section explique comment évaluer les paramètres de mise à l'échelle automatique de vos tables DynamoDB. La mise à l'échelle automatique Amazon DynamoDB est une fonctionnalité qui gère le débit des tables et des index secondaires globaux (GSI) en fonction du trafic de votre application et de votre métrique d'utilisation cible. Cela garantit vos tables ou GSIs disposera de la capacité requise pour vos modèles d'application.

Le service de dimensionnement AWS automatique surveillera l'utilisation actuelle de votre table et la comparera à la valeur d'utilisation cible :`TargetValue`. Il vous indique si vous devez augmenter ou diminuer la capacité allouée.

Rubriques

- Examen de vos paramètres de mise à l'échelle automatique
- Comment identifier les tables présentant une faible cible d'utilisation (<= 50 %)
- Comment gérer les charges de travail liées aux variations saisonnières
- Comment gérer les pics de charge de travail imprévisibles
- Comment gérer les charges de travail avec des applications liées

Examen de vos paramètres de mise à l'échelle automatique

Pour définir la valeur correcte correspondant à l'utilisation cible, ainsi que l'étape initiale et les valeurs finales, vous devez faire appel à l'équipe des opérations. Cela vous permet de définir de manière appropriée les valeurs en fonction de l'historique d'utilisation de l'application, valeurs qui seront utilisées pour déclencher les politiques de mise à l'échelle automatique AWS. La cible d'utilisation désigne le pourcentage de votre capacité totale qui doit être atteint pendant une période donnée avant que les règles de mise à l'échelle automatique ne s'appliquent.

Lorsque vous définissez une cible d'utilisation élevée (d'environ 90 %), le trafic doit dépasser 90 % pendant un certain temps, avant le démarrage de la mise à l'échelle automatique. Il est préférable de ne pas utiliser une cible d'utilisation élevée, sauf si votre application est très constante et ne fait l'objet d'aucun pic de trafic.

Lorsque vous définissez une cible d'utilisation très faible (inférieure à 50 %), votre application doit atteindre 50 % de la capacité allouée avant de déclencher une politique de mise à l'échelle automatique. À moins que le trafic de vos applications n'augmente à un rythme très soutenu, cela se traduit généralement par une capacité inutilisée et un gaspillage de ressources.

Comment identifier les tables présentant une faible cible d'utilisation (<= 50 %)

Vous pouvez utiliser le AWS CLI ou AWS Management Console pour surveiller et identifier les `TargetValues` politiques de dimensionnement automatique dans vos ressources DynamoDB :

AWS CLI

1. Pour afficher la liste complète des ressources, exécutez la commande suivante :

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   Cette commande renvoie la liste complète des politiques de mise à l'échelle automatique émises pour n'importe quelle ressource DynamoDB. Pour récupérer uniquement les

ressources d'une table particulière, vous pouvez ajouter le `–resource-id parameter`. Par exemple :

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>”
```

2. Renvoyez uniquement les politiques de mise à l'échelle automatique pour un GSI particulier en exécutant la commande suivante :

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

Les valeurs qui nous intéressent pour les politiques de mise à l'échelle automatique sont mises en évidence ci-dessous. Nous voulons nous assurer que la valeur cible est supérieure à 50 % afin d'éviter tout surprovisionnement. Le résultat doit ressembler à ce qui suit :

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
```

```
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

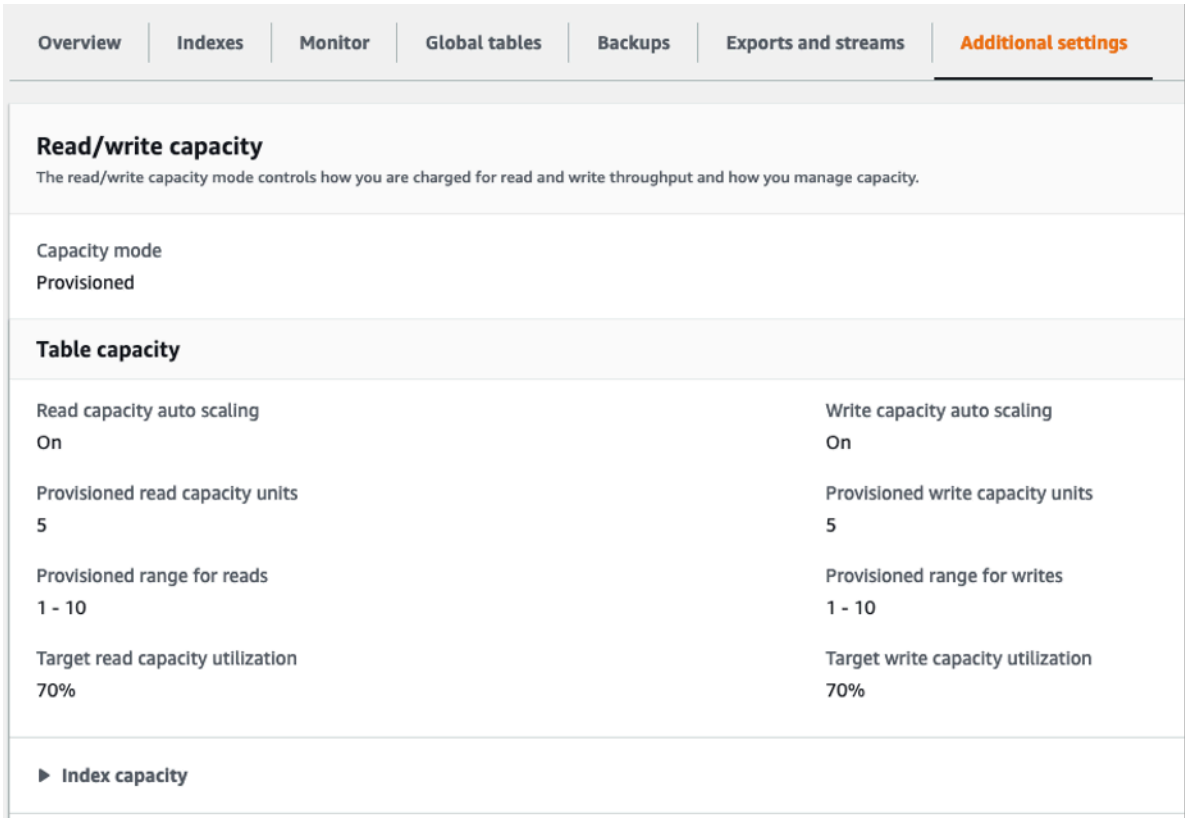
AWS Management Console

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/

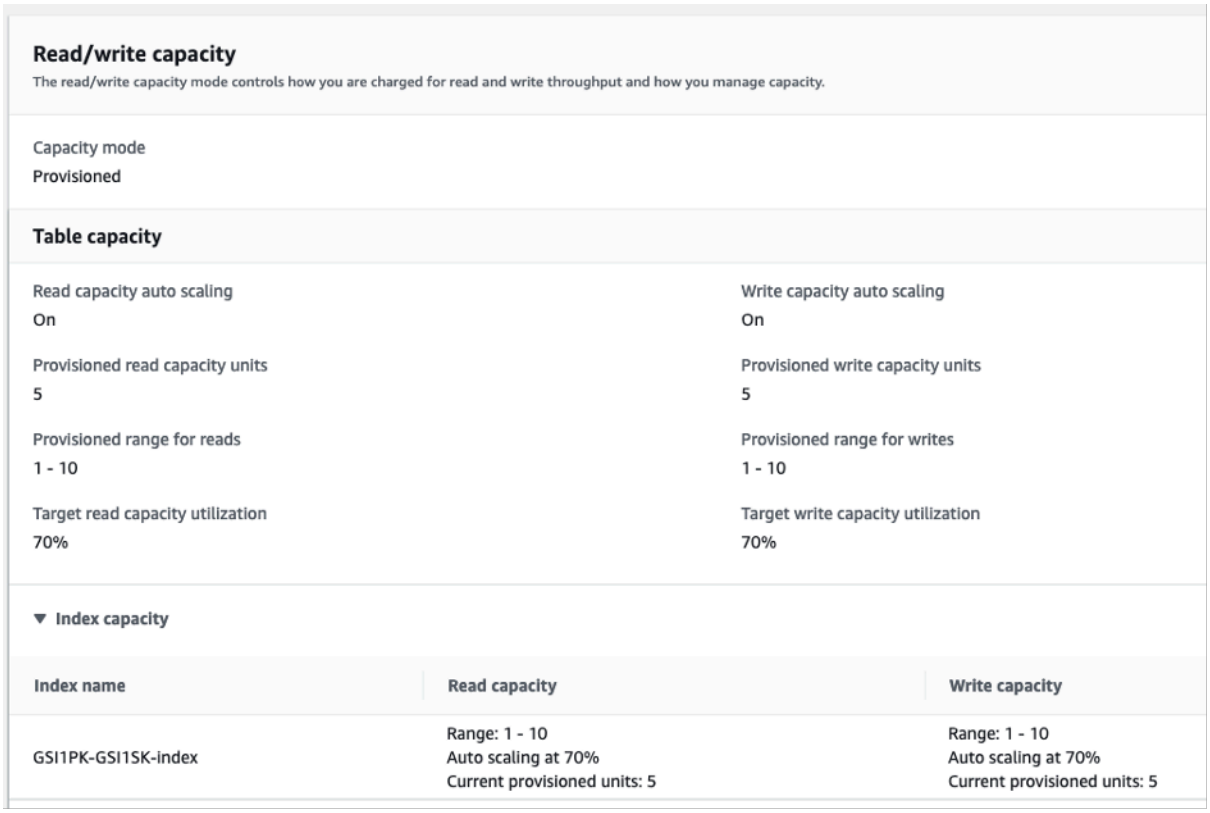
   Sélectionnez une option appropriée Région AWS si nécessaire.

2. Dans le volet de navigation, sélectionnez Tables. Sur la page Tables, sélectionnez le nom de la table.

3. Sur la page Détails de la table, sous l’onglet Paramètres supplémentaires, passez en revue les paramètres d’autoscaling de votre table.



Pour les index, ouvrez l'onglet Capacité de l'index pour consulter les paramètres d'autoscaling de l'index.



Si vos valeurs d'utilisation cibles sont inférieures ou égales à 50 %, explorez les métriques d'utilisation de vos tables pour déterminer si elles sont sous-provisionnées ou surprovisionnées.

Comment gérer les charges de travail liées aux variations saisonnières

Envisagez le scénario suivant : votre application fonctionne en dessous d'une valeur moyenne minimale la plupart du temps, mais la cible d'utilisation est faible. Votre application peut donc réagir rapidement aux événements qui se produisent à certaines heures de la journée et vous disposez d'une capacité suffisante pour éviter les ralentissements. Ce scénario est courant avec les applications qui sont très actives pendant les heures normales de bureau (de 9 h à 17 h), mais qui fonctionnent ensuite à un niveau de base en dehors de cette plage horaire. Comme certains utilisateurs commencent à se connecter avant 9 h, l'application utilise cette limite inférieure pour atteindre rapidement la capacité requise aux heures de pointe.

Ce scénario peut se présenter comme suit :

- Entre 17 h et 9 h, les unités `ConsumedWriteCapacity` restent entre 90 et 100.

- Les utilisateurs commencent à se connecter à l'application avant 9 heures du matin et les unités de capacité augmentent considérablement (la valeur maximale que vous avez vue est de 1 500 WCU).
- En moyenne, l'utilisation de vos applications varie entre 800 et 1 200 pendant les heures de travail.

Si le scénario précédent s'applique à votre cas de figure, envisagez d'utiliser une mise à l'échelle automatique planifiée. Une règle de mise à l'échelle automatique de votre application pourra toujours être configurée au niveau de votre table, mais avec une utilisation cible moins agressive qui n'allouera la capacité supplémentaire qu'aux intervalles spécifiques dont vous avez besoin.

Vous pouvez utiliser AWS CLI les étapes suivantes pour créer une règle de dimensionnement automatique planifiée qui s'exécutera en fonction de l'heure du jour et du jour de la semaine.

1. Enregistrez votre table DynamoDB ou votre index GSI en tant que cible pouvant être mise à l'échelle avec Application Auto Scaling. Une cible pouvant être mise à l'échelle avec est une ressource dont Application Auto Scaling peut augmenter ou réduire la capacité.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --min-capacity 90 \
    --max-capacity 1500
```

2. Configurez les actions planifiées en fonction de vos besoins.

   Deux règles sont nécessaires pour couvrir ce scénario : l'une pour augmenter la capacité et l'autre pour la réduire. Voici la première règle pour augmenter la capacité de l'action planifiée :

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-8-5-scheduled-action \
    --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
    --schedule "cron(45 8 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

   Voici la deuxième règle pour réduire la capacité de l'action planifiée :

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. Exécutez la commande suivante pour confirmer que les deux règles ont été activées :

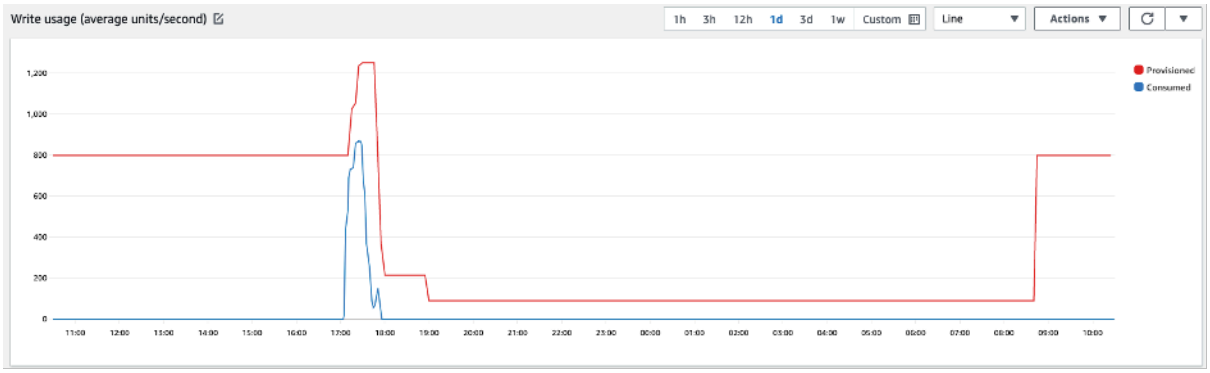
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Vous devriez obtenir le résultat suivant :

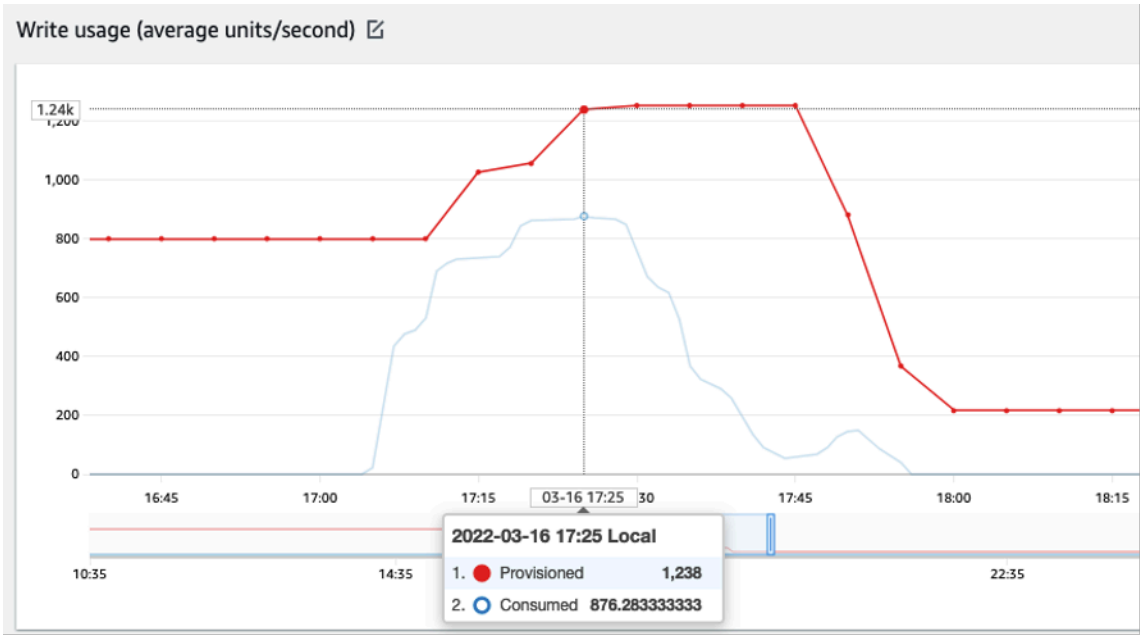
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
```

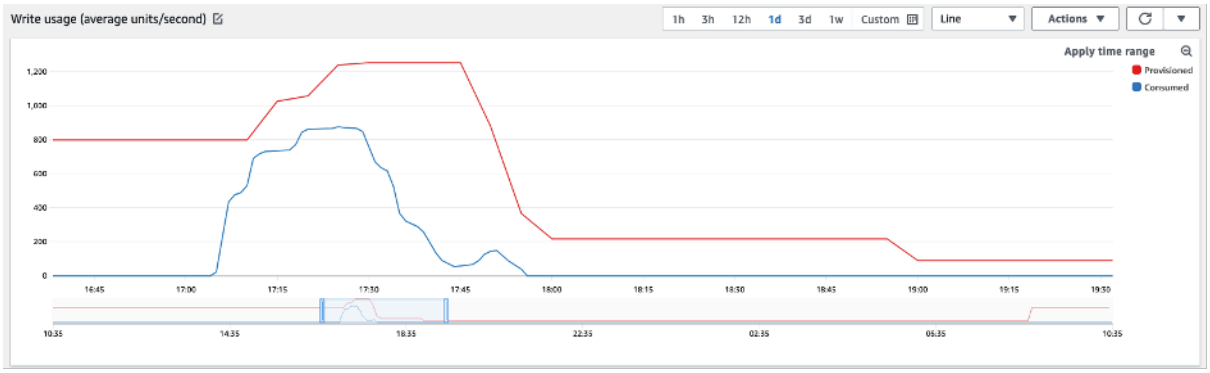
```
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

L'image suivante montre un exemple de charge de travail qui maintient en permanence une cible d'utilisation de 70 %. Notez que les règles de mise à l'échelle automatique s'appliquent toujours et que le débit n'est pas réduit.



En zoomant, nous pouvons constater qu'un pic dans l'application a déclenché le seuil de 70 %, forçant ainsi l'autoscaling à démarrer et à fournir la capacité supplémentaire requise pour la table. L'action de mise à l'échelle automatique planifiée affecte les valeurs maximales et minimales. C'est à vous qu'il revient de les configurer.





Comment gérer les pics de charge de travail imprévisibles

Dans ce scénario, l'application utilise une cible d'utilisation très faible, car l'évolution de son trafic n'est pas encore connue et vous souhaitez vous assurer que votre charge de travail n'est pas limitée.

Vous pouvez ici envisager d'utiliser le mode de capacité à la demande. Les tables à la demande sont idéales pour les charges de travail exigeantes pour lesquelles vous ne connaissez pas les tendances de trafic. Avec le mode de capacité à la demande, vous payez à la demande les lectures et écritures de données que votre application effectue sur vos tables. Vous n'avez pas besoin de spécifier le débit de lecture et d'écriture prévu pour votre application, car DynamoDB s'adapte instantanément à vos charges de travail à mesure qu'elles augmentent ou diminuent.

Comment gérer les charges de travail avec des applications liées

Dans ce scénario, l'application dépend d'autres systèmes, tels que les scénarios de traitement par lots dans lesquels vous pouvez avoir de forts pics de trafic en fonction des événements dans la logique de l'application.

Envisagez de développer une logique de mise à l'échelle automatique personnalisée qui réagit aux événements où vous pouvez augmenter la capacité de la table et `TargetValues` en fonction de vos besoins spécifiques. Vous pourriez bénéficier Amazon EventBridge et utiliser une combinaison de AWS services tels que Lambda et Step Functions pour répondre aux besoins spécifiques de votre application.

Évaluation de la sélection de votre classe de table DynamoDB

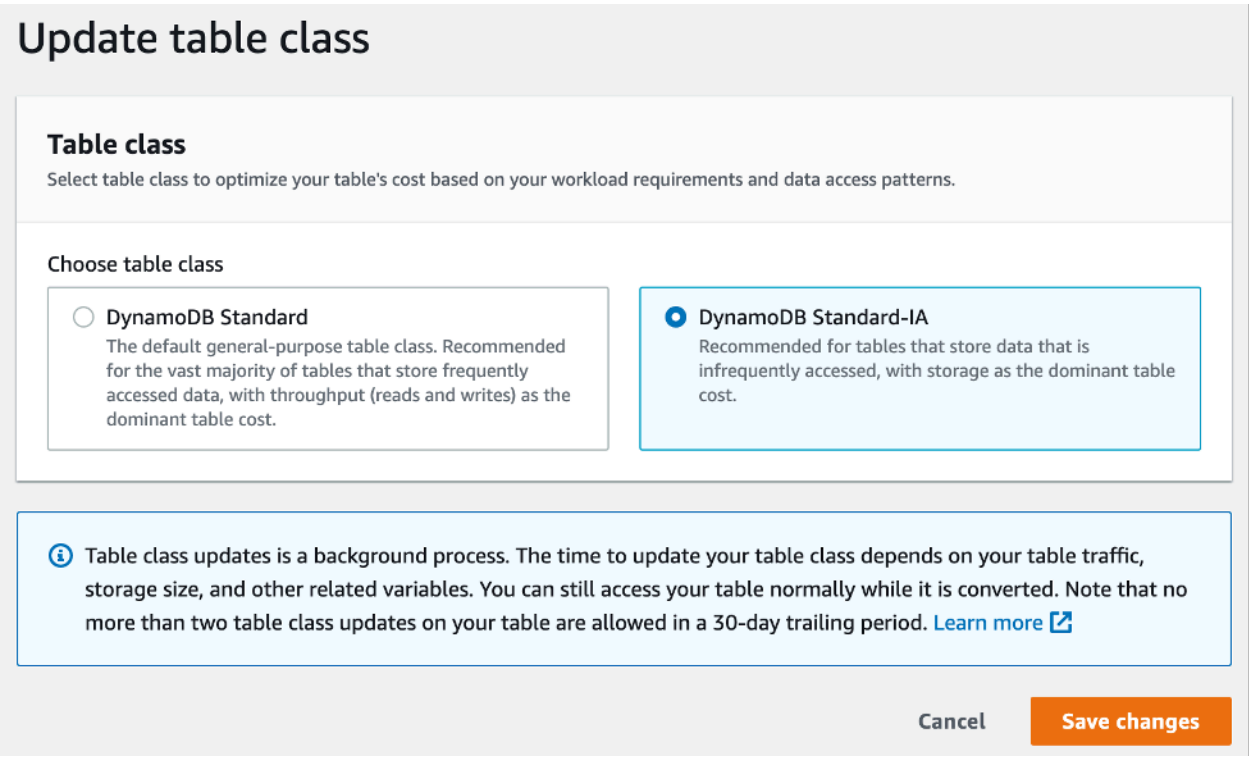
Cette section explique comment sélectionner la classe de table appropriée pour votre table DynamoDB. Avec le lancement de la classe de table Standard Infrequent-Access (Standard-IA), vous pouvez désormais optimiser une table de façon à réduire les coûts de stockage ou de débit.

Rubriques

- Quelles classes de table sont disponibles ?
- Quand sélectionner la classe de table DynamoDB Standard
- Quand sélectionner la classe de table DynamoDB Standard-IA
- Autres facteurs à prendre en considération lors du choix d'une classe de table

Quelles classes de table sont disponibles ?

Lorsque vous créez une table DynamoDB, vous devez sélectionner DynamoDB Standard ou DynamoDB Standard-IA pour la classe de table. La classe de table peut être modifiée deux fois sur une période de 30 jours, vous pourrez donc la modifier facilement si nécessaire à l'avenir. La sélection de l'une ou l'autre des classes de table n'a aucun effet sur les performances, la disponibilité, la fiabilité ou la durabilité de la table.



Classe de table Standard

La classe de table Standard est l'option par défaut pour les nouvelles tables. Avec cette option, vous pouvez conserver l'équilibre de facturation initial de DynamoDB, ce qui permet de trouver un équilibre entre le débit et les coûts de stockage pour les tables contenant des données fréquemment consultées.

Classe de table Standard-IA

La classe de tables Standard-IA offre un coût de stockage inférieur (d'environ 60 %) pour les charges de travail qui nécessitent un stockage à long terme des données avec des mises à jour ou des lectures peu fréquentes. Dans la mesure où la classe est optimisée pour les accès occasionnels, les lectures et les écritures seront facturées à un coût légèrement plus élevé (d'environ 25 %) que la classe de table Standard.

Quand sélectionner la classe de table DynamoDB Standard

La classe de table Standard de DynamoDB convient parfaitement aux tables dont le coût de stockage est inférieur ou égal à 50 % du coût mensuel global de la table. Cet équilibre des coûts indique une charge de travail qui accède régulièrement à des éléments déjà stockés dans DynamoDB ou qui les met à jour.

Quand sélectionner la classe de table DynamoDB Standard-IA

La classe de table Standard-IA de DynamoDB convient parfaitement aux tables dont le coût de stockage est d'environ 50 % ou plus du coût mensuel global de la table. Cet équilibre des coûts indique une charge de travail qui crée ou lit moins d'éléments par mois qu'elle n'en sauvegarde.

Une utilisation courante de la classe de tables Standard-IA consiste à déplacer les données moins fréquemment consultées vers des tables Standard-IA individuelles. Pour plus d'informations, consultez Optimisation des coûts de stockage de vos charges de travail avec la classe de table Amazon DynamoDB Standard-IA.

Autres facteurs à prendre en considération lors du choix d'une classe de table

Lorsque vous choisissez entre les deux classes de table, certains facteurs supplémentaires méritent d'être pris en compte dans le cadre de votre décision.

Capacité réservée

L'achat de capacité réservée pour les tables utilisant la classe de table Standard-IA n'est pas pris en charge pour l'instant. Lorsque vous passerez d'une table standard avec capacité réservée à une table standard IA sans capacité réservée, vous ne constaterez peut-être aucun avantage en termes de coûts.

Identification de vos ressources inutilisées dans DynamoDB

Cette section explique comment évaluer régulièrement vos ressources inutilisées. À mesure que les exigences de votre application évoluent, vous devez vous assurer qu'aucune ressource n'est inutilisée et ne contribue à des coûts inutiles pour Amazon DynamoDB. Les procédures décrites ci-dessous utiliseront les CloudWatch métriques Amazon pour identifier les ressources inutilisées et vous aideront à identifier ces ressources et à prendre les mesures nécessaires pour réduire les coûts.

Vous pouvez surveiller DynamoDB à CloudWatch l'aide de ce qui collecte et traite les données brutes de DynamoDB en métriques lisibles quasiment en temps réel. Ces statistiques étant conservées pendant un certain temps, vous pouvez accéder aux informations historiques pour acquérir une meilleure compréhension de votre utilisation. Par défaut, les données métriques DynamoDB sont envoyées automatiquement à. CloudWatch Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch ? et conservation des métriques dans le guide de CloudWatch l'utilisateur Amazon.

Rubriques

- Comment identifier les ressources inutilisées

- Identification des ressources de table inutilisées
- Nettoyage des ressources de table inutilisées
- Identification des ressources GSI inutilisées
- Nettoyage des ressources GSI inutilisées
- Nettoyage des tables globales inutilisées
- Nettoyage des sauvegardes ou point-in-time des restaurations non utilisées (PITR)

Comment identifier les ressources inutilisées

Pour identifier les tables ou les index inutilisés, nous examinerons les CloudWatch indicateurs suivants sur une période de 30 jours afin de déterminer s'il existe des lectures ou des écritures actives sur la table ou des lectures sur les index secondaires globaux () GSIs :

ConsumedReadCapacityUnits

Nombre d’unités de capacité de lecture consommées sur la période spécifiée, de sorte que vous puissiez suivre la capacité consommée. Vous pouvez extraire la capacité totale de lecture consommée pour une table et tous ses index secondaires globaux, ou pour un index secondaire global particulier.

ConsumedWriteCapacityUnits

Nombre d’unités de capacité d’écriture consommées sur la période spécifiée, de sorte que vous puissiez suivre la capacité consommée. Vous pouvez extraire la capacité totale d’écriture consommée pour une table et tous ses index secondaires globaux, ou pour un index secondaire global particulier.

Identification des ressources de table inutilisées

Amazon CloudWatch est un service de surveillance et d'observabilité qui fournit les métriques des tables DynamoDB que vous utiliserez pour identifier les ressources inutilisées. CloudWatch les métriques peuvent être consultées àAWS Management Console la fois par le biais du AWS Command Line Interface.

AWS Command Line Interface

Pour afficher les statistiques de vos tables via le AWS Command Line Interface, vous pouvez utiliser les commandes suivantes.

1. Tout d’abord, évaluez les lectures de votre table :

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Pour éviter de faussement identifier une table comme étant inutilisée, évaluez les indicateurs sur une période plus longue. Choisissez une plage d'heure de début et de fin appropriée, par exemple 30 jours, et une période appropriée, telle que 86400.

Dans les données renvoyées, toute somme supérieure à 0 indique que la table que vous évaluez a reçu du trafic de lecture pendant cette période.

Le résultat suivant montre une table recevant du trafic de lecture au cours de la période évaluée :

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

Le résultat suivant montre qu'une table n'a pas reçu de trafic de lecture au cours de la période évaluée :

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. Ensuite, évaluez les écritures dans votre table :

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Pour éviter de faussement identifier une table comme étant inutilisée, vous devez évaluer les indicateurs sur une période plus longue. Choisissez une plage d'heure de début et de fin appropriée, par exemple 30 jours, et une période appropriée, telle que 86400.

Dans les données renvoyées, toute somme supérieure à 0 indique que la table que vous évaluez a reçu du trafic de lecture pendant cette période.

Le résultat suivant montre une table recevant du trafic d'écriture au cours de la période évaluée :

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

Le résultat suivant montre qu'une table n'a pas reçu de trafic d'écriture au cours de la période évaluée :

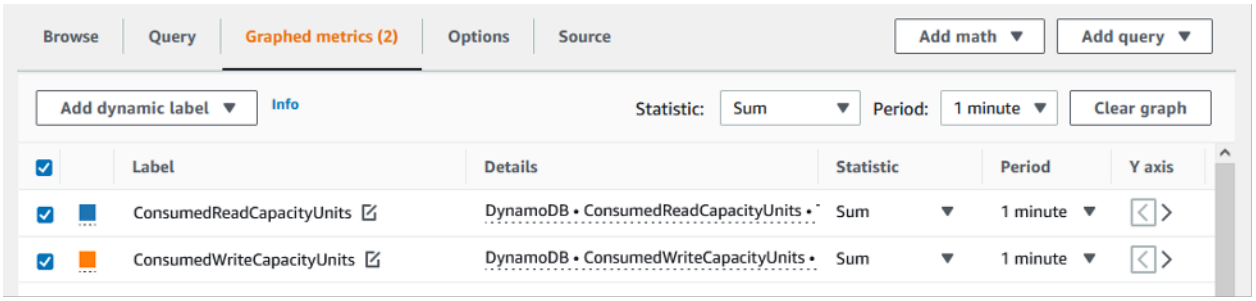
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
```

```
        },
```

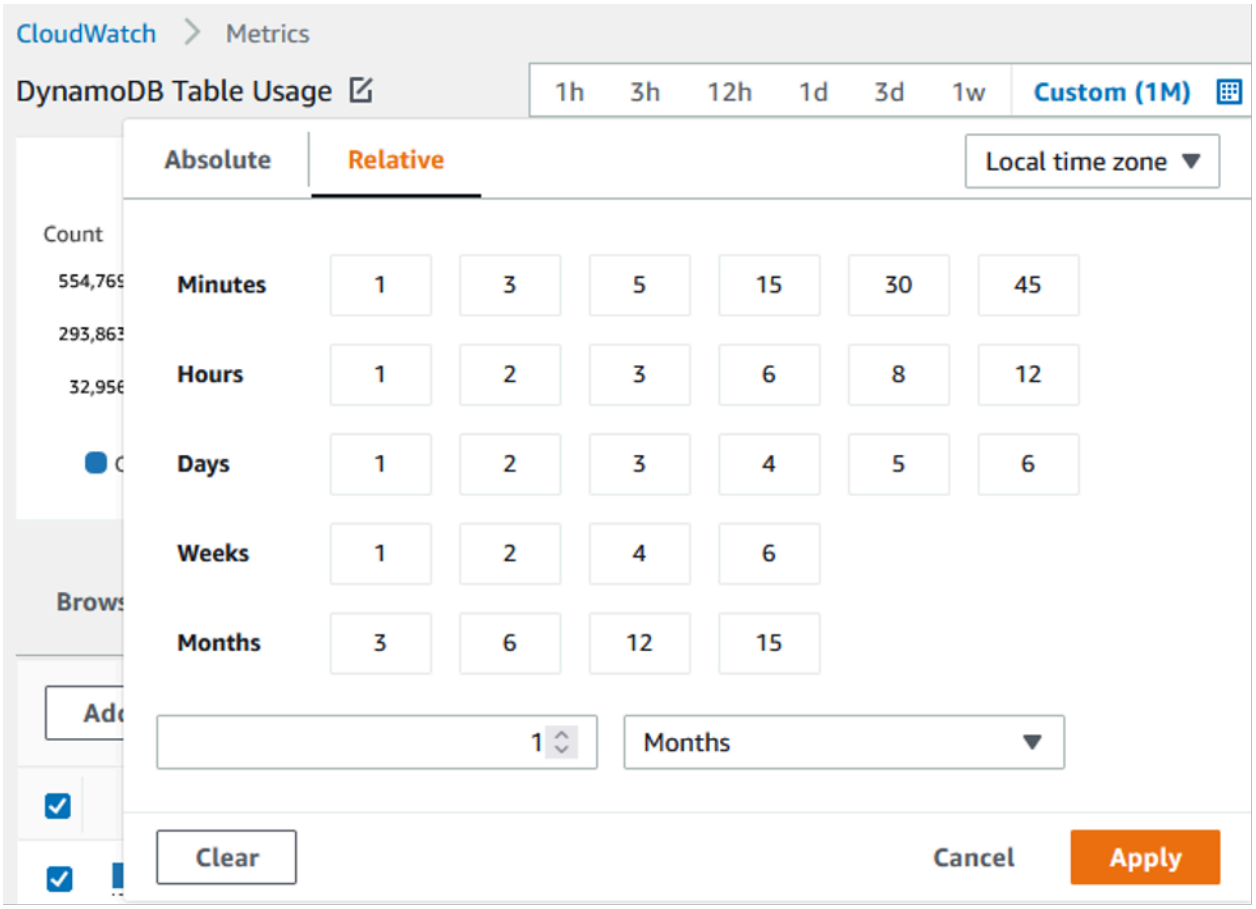
AWS Management Console

Les étapes suivantes vous permettront d'évaluer l'utilisation de vos ressources via la AWS Management Console.

1. Connectez-vous à la AWS console et accédez à la page de CloudWatch service à l'adresse https://console.aws.amazon.com/cloudwatch/. Sélectionnez la AWS région appropriée en haut à droite de la console, si nécessaire.
2. Dans la barre de navigation de gauche, sélectionnez la section Metrics (Métriques), puis sélectionnez All metrics (Toutes les métriques).
3. L'action ci-dessus ouvrira un tableau de bord avec deux panneaux. Dans le panneau supérieur, vous verrez les métriques actuellement représentées sous forme de graphique. En bas, vous allez sélectionner les métriques disponibles pour une représentation graphique. Sélectionnez DynamoDB dans le panneau inférieur.
4. Dans le panneau de sélection des métriques de DynamoDB, sélectionnez la catégorie Table Metrics (Métriques de table) pour afficher les métriques de vos tables dans la région actuelle.
5. Identifiez le nom de votre table en faisant défiler le menu vers le bas, puis sélectionnez les métriques `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits` pour votre table.
6. Sélectionnez l'onglet Graphed metrics (2) (Métriques représentées graphiquement) et définissez la colonne Statistics (Statistiques) sur Sum (Somme).

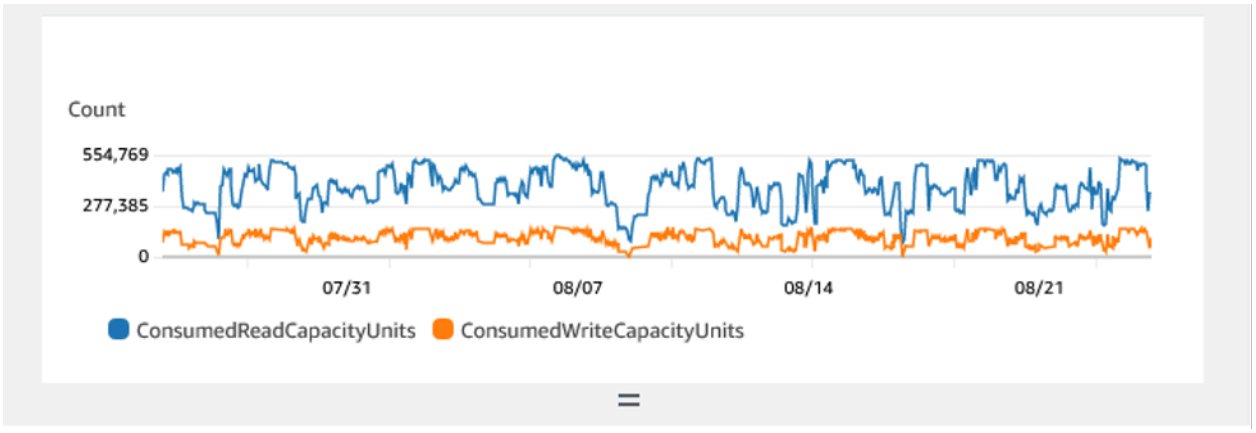


7. Pour éviter de faussement identifier une table comme étant inutilisée, vous devez évaluer les indicateurs sur une période plus longue. En haut du panneau graphique, choisissez une période appropriée, par exemple 1 mois, pour évaluer votre table. Sélectionnez Custom (Personnalisé), sélectionnez 1 Months (1 mois) dans les listes déroulantes, puis choisissez Apply (Appliquer).

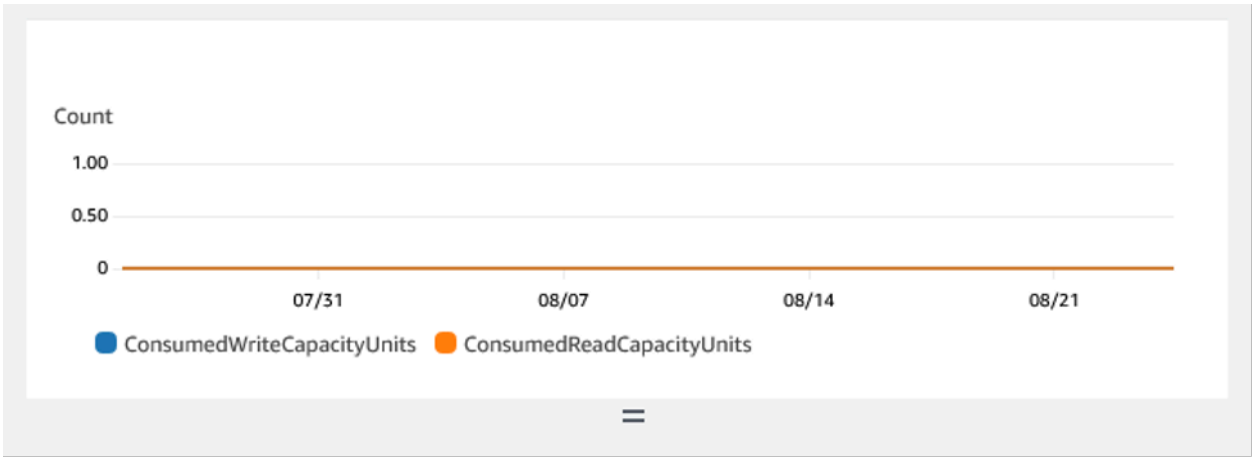


8. Évaluez les métriques représentées graphiquement pour votre table afin de déterminer si elle est utilisée. Les métriques supérieures à 0 indiquent qu'une table a été utilisée pendant la période évaluée. Un graphique plat à 0 pour la lecture et l'écriture indique une table qui n'est pas utilisée.

   L'image suivante montre une table avec le trafic de lecture :



   L'image suivante montre une table sans trafic de lecture.



Nettoyage des ressources de table inutilisées

Si vous avez identifié des ressources de table inutilisées, vous pouvez réduire leurs coûts permanents de la manière suivante.

> **Note**
>
> Si vous avez identifié une table non utilisée mais que vous souhaitez la garder à disposition pour un accès futur, pensez à la passer en mode à la demande. Sinon, vous pouvez envisager de sauvegarder et de supprimer entièrement la table.

Modes de capacité

DynamoDB facture la lecture, l'écriture et le stockage de données dans vos tables DynamoDB.

DynamoDB propose deux modes de capacité, qui offrent des options de facturation spécifiques pour traiter les lectures et écritures dans vos tables. Le mode read/write capacité contrôle la façon dont vous êtes facturé pour le débit de lecture et d'écriture et la façon dont vous gérez la capacité.

Pour les tables en mode à la demande, vous devez spécifier le débit de lecture et d'écriture que votre application est supposée atteindre. DynamoDB facture les lectures et écritures que votre application effectue dans vos tables, comptabilisées en termes d'unités de demande de lecture et d'unités de demande d'écriture. S'il n'y a aucune activité sur votre ordinateur, table/index vous ne payez pas pour le débit, mais vous devrez tout de même payer des frais de stockage.

Classe de table

DynamoDB propose deux classes de tables conçues pour vous aider à optimiser vos coûts. La classe de tables DynamoDB Standard est la classe par défaut. Elle est recommandée pour la plupart des charges de travail. La classe de tables DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) est optimisée pour les tables où le stockage est le coût dominant.

S'il n'y a aucune activité sur votre table ou votre index, le stockage sera probablement le principal coût et le changement de classe de table permettra de réaliser des économies importantes.

Suppression des tables

Si vous avez découvert une table inutilisée et que vous souhaitez la supprimer, vous pouvez d'abord effectuer une sauvegarde ou une exportation des données.

Les sauvegardes effectuées via AWS Backup peuvent tirer parti de la hiérarchisation du stockage à froid, ce qui permet de réduire encore les coûts. Reportez-vous à la Utilisation de AWS Backup avec DynamoDB documentation pour savoir comment activer les sauvegardes via AWS Backup, ainsi qu'à la documentation sur la gestion des plans de sauvegarde pour savoir comment utiliser le cycle de vie pour transférer votre sauvegarde vers un stockage à froid.

Vous pouvez également choisir d'exporter les données de votre table vers S3. Pour ce faire, consultez la documentation relative à l'exportation vers Amazon S3. Une fois vos données exportées, si vous souhaitez tirer parti de S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval ou S3 Glacier Deep Archive afin de réduire davantage les coûts, consultez Gestion de votre cycle de vie de stockage.

Une fois votre table sauvegardée, vous pouvez choisir de la supprimer via la AWS Management Console ou via l'AWS Command Line Interface.

Identification des ressources GSI inutilisées

Les étapes d'identification d'un secondaire global non utilisé sont similaires à celles d'identification d'une table non utilisée. Étant donné que DynamoDB réplique les éléments écrits dans votre table de base dans votre GSI s'ils contiennent l'attribut utilisé comme clé de partition du GSI, un GSI inutilisé est toujours susceptible d'avoir une valeur `ConsumedWriteCapacityUnits` supérieure à 0 si sa table de base est utilisée. Par conséquent, vous n'évaluerez que la métrique `ConsumedReadCapacityUnits` pour déterminer si votre GSI n'est pas utilisé.

Pour afficher vos métriques GSI via le AWSAWS CLI, vous pouvez utiliser les commandes suivantes pour évaluer les lectures de vos tables :

```
aws cloudwatch get-metric-statistics --metric-name
```

```
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

Pour éviter de faussement identifier une table comme étant inutilisée, vous devez évaluer les indicateurs sur une période plus longue. Choisissez une plage d'heure de début et de fin appropriée, par exemple 30 jours, et une période appropriée, telle que 86400.

Dans les données renvoyées, toute somme supérieure à 0 indique que la table que vous évaluez a reçu du trafic de lecture pendant cette période.

Le résultat suivant montre qu'un GSI a reçu du trafic de lecture au cours de la période évaluée :

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

Le résultat suivant montre qu'un GSI reçoit un trafic de lecture minimal au cours de la période évaluée :

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

Le résultat suivant montre qu'un GSI n'a reçu aucun trafic de lecture au cours de la période évaluée :

```
        {
```

```
        "Timestamp": "2022-08-17T21:20:00Z",
        "Sum": 0.0,
        "Unit": "Count"
      },
      {
        "Timestamp": "2022-08-11T21:20:00Z",
        "Sum": 0.0,
        "Unit": "Count"
      },
```

Nettoyage des ressources GSI inutilisées

Si vous avez identifié un GSI non utilisé, vous pouvez choisir de le supprimer. Comme toutes les données présentes dans un GSI sont également présentes dans la table de base, aucune sauvegarde supplémentaire n'est nécessaire avant de supprimer un GSI. Si, à l'avenir, le GSI est à nouveau nécessaire, il pourra être ajouté à nouveau à la table.

Si vous avez identifié un GSI peu utilisé, vous devez envisager de modifier la conception de votre application afin de le supprimer ou de réduire son coût. Par exemple, si les analyses DynamoDB doivent être utilisées avec parcimonie car elles peuvent consommer de grandes quantités de ressources système, elles peuvent s'avérer plus rentables qu'un GSI si le modèle d'accès pris en charge est très peu utilisé.

De plus, si un GSI est requis pour prendre en charge un modèle d'accès peu fréquent, envisagez de projeter un ensemble plus limité d'attributs. Bien que cela puisse nécessiter des requêtes ultérieures sur la table de base pour prendre en charge vos modèles d'accès peu fréquents, cela peut potentiellement permettre de réduire considérablement les coûts de stockage et d'écriture.

Nettoyage des tables globales inutilisées

Les tables globales Amazon DynamoDB fournissent une solution entièrement gérée pour déployer une base de données multiactive et multirégion sans devoir créer et gérer votre propre solution de réplication.

Les tables globales sont idéales pour fournir un accès à faible latence aux données à proximité des utilisateurs et constituent une région secondaire pour la reprise après sinistre.

Si l'option des tables globales est activée pour une ressource afin de fournir un accès à faible latence aux données, mais que cela ne fait pas partie de votre stratégie de reprise après sinistre, vérifiez que les deux répliques desservent activement le trafic de lecture en évaluant leurs CloudWatch indicateurs. Si un réplica ne sert pas le trafic de lecture, il s'agit peut-être d'une ressource inutilisée.

Si les tables globales font partie de votre stratégie de reprise après sinistre, on peut s'attendre à ce qu'une réplique ne reçoive pas de trafic de lecture selon un active/standby schéma.

Nettoyage des sauvegardes ou point-in-time des restaurations non utilisées (PITR)

DynamoDB propose deux styles de sauvegarde. Point-in-timeLa restauration fournit des sauvegardes continues pendant 35 jours maximum pour vous protéger contre les écritures ou les suppressions accidentelles, tandis que la sauvegarde à la demande permet de créer des instantanés qui peuvent être enregistrés à long terme. Vous pouvez définir la période de restauration sur une valeur comprise entre 1 et 35 jours. Les deux types de sauvegardes sont associés à des coûts.

Reportez-vous à la documentation Sauvegarde et restauration pour DynamoDB et Sauvegardes ponctuelles pour DynamoDB afin de déterminer si vos tables disposent de sauvegardes activées qui ne sont peut-être plus nécessaires.

Évaluer les modèles d'utilisation de votre table DynamoDB

Cette section explique comment déterminer si vous utilisez efficacement votre table DynamoDB. Certains modèles d'utilisation ne sont pas optimaux pour DynamoDB et peuvent être optimisés du point de vue des performances et des coûts.

Rubriques

- Effectuer moins d'opérations de lecture fortement cohérente
- Réaliser moins de transactions pour les opérations de lecture
- Effectuer moins d'analyses
- Raccourcir le nom des attributs
- Activer la durée de vie (TTL)
- Remplacer les tables globales par des sauvegardes interrégionales

Effectuer moins d'opérations de lecture fortement cohérente

DynamoDB vous permet de configurer la cohérence de lecture pour chaque demande. Les demandes de lecture sont « éventuellement cohérentes » par défaut. Ces lectures sont facturées à 0,5 RCU pour un maximum de 4 Ko de données.

La plupart des éléments des charges de travail distribuées sont flexibles et peuvent tolérer une éventuelle cohérence. Cependant, certains modèles d'accès peuvent nécessiter des lectures

fortement cohérentes. Les lectures fortement cohérentes sont facturées à 1 RCU pour un maximum de 4 Ko de données, ce qui multiplie les coûts de lecture par deux. DynamoDB vous offre la flexibilité nécessaire pour utiliser les deux modèles de cohérence au niveau de la même table.

Vous pouvez évaluer votre charge de travail et le code de votre application pour vérifier si les lectures fortement cohérentes ne sont utilisées que lorsque cela est nécessaire.

Réaliser moins de transactions pour les opérations de lecture

DynamoDB vous permet de regrouper certaines actions d' all-or-nothingune certaine manière, ce qui signifie que vous pouvez exécuter des transactions ACID avec DynamoDB. Toutefois, comme c'est le cas pour les bases de données relationnelles, il n'est pas nécessaire de suivre cette approche pour chaque action.

Une opération de lecture transactionnelle allant jusqu'à 4 Ko RCUs en consomme 2, contre 0,5 par défaut RCUs pour lire la même quantité de données de manière finalement cohérente. Les coûts des opérations d'écriture sont doublés, ce qui signifie qu'une écriture transactionnelle allant jusqu'à 1 Ko équivaut à 2. WCUs

Pour déterminer si toutes les opérations de vos tables sont des transactions, CloudWatch les mesures de la table peuvent être filtrées en fonction de la transaction APIs. Si APIs les transactions sont les seuls graphiques disponibles sous les `SuccessfulRequestLatency` métriques du tableau, cela confirmera que chaque opération est une transaction pour ce tableau. Sinon, si la tendance globale d'utilisation des capacités correspond à la tendance des API transactionnelles, pensez à revoir la conception de l'application, car elle semble dominée par les transactions APIs.

Effectuer moins d'analyses

L'utilisation intensive des opérations `Scan` est généralement liée à la nécessité d'exécuter des requêtes analytiques au niveau des données DynamoDB. L'exécution d'opérations `Scan` fréquentes sur une table de grande envergure peut s'avérer inefficace et coûteuse.

Une meilleure alternative consiste à utiliser la fonctionnalité Exporter vers S3 et à choisir un moment pour exporter l'état de la table vers S3.AWS propose des services tels qu'Athena, qui peuvent ensuite être utilisés pour exécuter des requêtes analytiques sur les données sans consommer la capacité de la table.

La fréquence des opérations `Scan` peut être déterminée à l'aide de la statistique `SampleCount` sous la métrique `SuccessfulRequestLatency` pour `Scan`. Si les opérations `Scan` sont très fréquentes, les modèles d'accès et le modèle de données devraient être réévalués.

Raccourcir le nom des attributs

La taille totale d'un élément dans DynamoDB est la somme des longueurs de ses noms et valeurs d'attribut. Le fait d'avoir des noms d'attributs longs contribue non seulement aux coûts de stockage, mais peut également entraîner une augmentation de RCU/WCU la consommation. Nous vous recommandons de choisir des noms d'attribut courts. Le fait d'avoir des noms d'attributs plus courts peut aider à limiter la taille de l'élément dans la prochaine limite de 4 Ko/1 Ko, après quoi vous consommerez davantage RCU/WCU de données d'accès.

Activer la durée de vie (TTL)

La durée de vie (TTL) permet d'identifier les éléments dont la date d'expiration est supérieure à celle que vous avez définie, et de les supprimer de la table. Si vos données augmentent au fil du temps et que des données plus anciennes ne sont plus pertinentes, l'activation du TTL au niveau de la table contribue à réduire la quantité de données et à diminuer les coûts de stockage.

Un autre aspect utile du TTL est que les éléments ayant expiré se trouvent dans vos flux DynamoDB. Par conséquent, au lieu de simplement supprimer les données obsolètes parmi toutes vos données, il est possible de consommer ces éléments du flux et de les archiver vers un niveau de stockage moins coûteux. En outre, la suppression d'éléments via le TTL n'entraîne aucun coût supplémentaire : il ne consomme pas de capacité, et il n'est pas nécessaire de passer un temps précieux à concevoir une application de nettoyage.

Remplacer les tables globales par des sauvegardes interrégionales

Les tables globales vous permettent de gérer plusieurs tables de réplica actives dans différentes régions. Elles peuvent toutes accepter des opérations d'écriture et répliquer des données entre elles. Il est facile de configurer des réplicas. La synchronisation est gérée pour vous. Les réplicas convergent vers un état cohérent grâce à une stratégie de type « priorité à la dernière écriture ».

Si vous utilisez des tables globales uniquement dans le cadre d'une stratégie de basculement ou de reprise après sinistre, vous pouvez les remplacer par une copie de sauvegarde interrégionale pour des objectifs de points de restauration et des exigences en matière de temps de restauration relativement modérés. Si vous n'avez pas besoin d'un accès local rapide ni d'une haute disponibilité de 99,999 %, la gestion d'un réplica de table globale n'est peut-être pas la meilleure approche pour le basculement.

Vous pouvez également envisager d'utiliser AWS Backup pour gérer les sauvegardes DynamoDB. Vous pouvez planifier des sauvegardes régulières et les copier entre les régions afin de répondre aux exigences de reprise après sinistre, d'une manière plus rentable que l'utilisation de tables globales.

Évaluation de l’utilisation de vos flux DynamoDB

Cette section explique comment évaluer votre utilisation de DynamoDB Streams. Certains modèles d’utilisation ne sont pas optimaux pour DynamoDB et peuvent être optimisés du point de vue des performances et des coûts.

Vous disposez de deux intégrations natives pour le streaming et les cas d’utilisation pilotés par des événements :

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Cette page se concentre uniquement sur les stratégies d’optimisation des coûts pour ces options. Si vous souhaitez découvrir comment faire un choix entre ces deux options, consultez Options de streaming pour la récupération de données de modification.

Rubriques

- Optimisation des coûts pour DynamoDB Streams
- Optimisation des coûts pour Kinesis Data Streams
- Stratégies d’optimisation des coûts pour les deux types d’utilisation de Streams

Optimisation des coûts pour DynamoDB Streams

Comme indiqué sur la page de tarification de DynamoDB Streams, quel que soit le mode de capacité de débit de la table, DynamoDB facture le nombre de demandes de lecture effectuées vers le flux DynamoDB de la table. Les demandes de lecture adressées à un flux DynamoDB sont différentes des demandes de lecture adressées à une table DynamoDB.

Chaque demande de lecture en termes de flux se présente sous la forme d’un appel d’API `GetRecords` qui peut renvoyer jusqu’à 1 000 enregistrements ou 1 Mo d’enregistrements dans la réponse, selon la première éventualité. Aucun des autres flux DynamoDB n'est chargé et aucun APIs flux DynamoDB n'est facturé en cas d'inactivité. En d’autres termes, si aucune demande de lecture n’est envoyée à un flux DynamoDB, aucuns frais ne seront facturés pour ce flux, même s’il est activé.

Voici quelques applications consommateur pour DynamoDB Streams :

- AWS Lambda fonction (s)
- Applications basées sur Amazon Kinesis Data Streams

- Applications destinées aux clients et aux consommateurs créées à l'aide d'un AWS SDK

Les demandes de lecture effectuées par les utilisateurs de DynamoDB Streams AWS basés sur Lambda sont gratuites, tandis que les appels effectués par des consommateurs de tout autre type sont payants. Chaque mois, les 2 500 000 premières demandes de lecture effectuées par des consommateurs autres que Lambda sont également gratuites. Cela s'applique à toutes les demandes de lecture adressées à un DynamoDB Streams dans un AWS compte pour chaque région.AWS

Surveillance de l’utilisation de DynamoDB Streams

Les frais DynamoDB Streams sur la console de facturation sont regroupés pour tous les flux DynamoDB de la région dans un compte.AWSAWS Actuellement, le balisage des flux DynamoDB n’est pas pris en charge. Les balises de répartition des coûts ne peuvent donc pas être utilisées pour identifier les coûts précis de DynamoDB Streams. Le volume d’appels `GetRecords` peut être obtenu au niveau du flux DynamoDB pour calculer les frais par flux. Le volume est représenté par la `SuccessfulRequestLatency` métrique et les statistiques CloudWatch du flux DynamoDB. `SampleCount` Cette métrique inclura également les `GetRecords` appels effectués par les tables globales pour effectuer une réplication continue ainsi que les appels passés par les clients AWS Lambda, tous deux gratuits. Pour plus d'informations sur CloudWatch les autres métriques publiées par DynamoDB Streams, consultez. Métriques et dimensions DynamoDB

Utiliser AWS Lambda en tant que consommateur

Évaluez s'il est possible d'utiliser les fonctions AWS Lambda comme consommateurs pour les flux DynamoDB, car cela peut éliminer les coûts associés à la lecture depuis le flux DynamoDB. En revanche, les applications consommateur basées sur le SDK ou l’adaptateur DynamoDB Streams Kinesis sont facturées en fonction du nombre d’appels `GetRecords` qu’elles effectuent vers le flux DynamoDB.

Les appels de fonctions Lambda sont facturés sur la base de la tarification Lambda standard, mais DynamoDB Streams n’entraîne pas de frais. Lambda interroge les partitions du flux DynamoDB en quête d’enregistrements à une fréquence de base de quatre fois par seconde. Lorsque des enregistrements sont disponibles, Lambda invoque votre fonction et attend le résultat. Si le traitement réussit, Lambda reprend l’interrogation jusqu’à recevoir plus d’enregistrements.

Optimisation des applications consommateur basées sur l’adaptateur DynamoDB Streams Kinesis

Étant donné que les demandes de lecture effectuées par les consommateurs n’utilisant pas Lambda sont facturées pour DynamoDB Streams, il est important de trouver un juste milieu entre les

exigences de traitement en temps quasi réel et le nombre de fois que l'application consommateur doit interroger le flux DynamoDB.

La fréquence d'interrogation de DynamoDB Streams à l'aide d'une application basée sur un adaptateur DynamoDB Streams Kinesis est déterminée par la valeur `idleTimeBetweenReadsInMillis` configurée. Ce paramètre détermine la durée en millisecondes pendant laquelle le consommateur doit attendre avant de traiter une partition au cas où l'appel `GetRecords` précédent effectué vers la même partition n'aurait renvoyé aucun enregistrement. Par défaut, la valeur de ce paramètre est de 1 000 ms. Si le traitement en temps quasi réel n'est pas requis, la valeur de ce paramètre peut être augmentée pour que l'application consommateur effectue moins d'appels `GetRecords` et optimise les appels DynamoDB Streams.

Optimisation des coûts pour Kinesis Data Streams

Lorsqu'un flux de données Kinesis est défini comme destination pour fournir des événements de récupération des données de modification pour une table DynamoDB, il peut nécessiter une gestion de son dimensionnement distincte, ce qui a une incidence sur les coûts globaux. DynamoDB facture en termes d'unités de capture des données de modification CDUs (), chaque unité étant composée d'un élément DynamoDB de 1 Ko essayé par le service DynamoDB vers le flux de données Kinesis de destination.

Outre les frais liés au service DynamoDB, des frais standard sont facturés pour le flux de données Kinesis. Comme indiqué sur la page de tarification, la tarification du service varie en fonction du mode de capacité (provisionné et à la demande), qui sont distincts des modes de capacité des tables DynamoDB et qui sont définis par l'utilisateur. De manière générale, Kinesis Data Streams facture un taux horaire basé sur le mode de capacité, ainsi que sur les données ingérées dans le flux par le service DynamoDB. Des frais supplémentaires peuvent être facturés, tels que la récupération de données (pour le mode à la demande), la conservation prolongée des données (au-delà des 24 heures par défaut) et la récupération améliorée par les consommateurs, en fonction de la configuration utilisateur du flux de données Kinesis.

Surveillance de l'utilisation de Kinesis Data Streams

Kinesis Data Streams for DynamoDB publie des métriques issues de DynamoDB en plus des métriques Kinesis Data Stream standard. CloudWatch Il est possible qu'une `Put` tentative du service DynamoDB soit bloquée par le service Kinesis en raison d'une capacité insuffisante de Kinesis Data Streams, ou par des composants dépendants AWS KMS tels qu'un service configuré pour chiffrer les données Kinesis Data Stream au repos.

Pour en savoir plus sur CloudWatch les métriques publiées par le service DynamoDB pour le Kinesis Data Stream, consultez. Surveiller la récupération de données de modification avec Kinesis Data Streams Afin d'éviter les coûts supplémentaires liés aux nouvelles tentatives de service dues à des limitations, il est important de dimensionner correctement le flux de données Kinesis en mode provisionné.

Choix du mode de capacité approprié pour Kinesis Data Streams

Les flux de données Kinesis proposent deux modes de capacité : le mode provisionné et le mode à la demande.

- Si la charge de travail impliquant un flux de données Kinesis génère un trafic d'application prévisible, un trafic constant ou en augmentation progressive, ou un trafic pouvant être prévu avec précision, le mode provisionné de Kinesis Data Streams est préférable et sera plus rentable.
- Si la charge de travail est nouvelle, si le trafic d'application est imprévisible ou si vous préférez ne pas gérer la capacité, le mode à la demande de Kinesis Data Streams est préférable et sera plus rentable.

Pour optimiser les coûts, il est recommandé d'évaluer si la table DynamoDB associée au flux de données Kinesis présente un modèle de trafic prévisible capable de tirer parti du mode provisionné de Kinesis Data Streams. S'il s'agit d'une nouvelle charge de travail, vous pouvez utiliser le mode à la demande pour les Kinesis Data Streams pendant les premières semaines, passer en revue CloudWatch les indicateurs pour comprendre les modèles de trafic, puis passer le même flux en mode provisionné en fonction de la nature de la charge de travail. En mode provisionné, l'estimation du nombre de partitions peut être effectuée en suivant les considérations de gestion des partitions pour Kinesis Data Streams.

Évaluer les applications consommateur à l'aide de Kinesis Data Streams pour DynamoDB

Comme Kinesis Data Streams ne facture pas en fonction du nombre d'appels `GetRecords`, comme c'est le cas pour DynamoDB Streams, les applications consommateur peuvent effectuer autant d'appels que possible, à condition que la fréquence soit inférieure aux limitations pour `GetRecords`. En ce qui concerne le mode à la demande pour Kinesis Data Streams, les lectures de données sont facturées au Go. Pour les flux de données Kinesis en mode provisionné, les lectures ne sont pas facturées si les données datent de moins de 7 jours. Dans le cas des fonctions Lambda en tant que consommateurs Kinesis Data Streams, Lambda interroge chaque partition de votre flux Kinesis en quête d'enregistrements à une fréquence de base d'une fois par seconde.

Stratégies d'optimisation des coûts pour les deux types d'utilisation de Streams

Filtrage des événements pour les utilisateurs de AWS Lambda

Le filtrage des événements Lambda vous permet de supprimer des évènements du lot d'appel de la fonction Lambda en fonction d'un critère de filtre. Cette approche permet d'optimiser les coûts Lambda liés au traitement ou à la suppression des enregistrements de flux indésirables dans le cadre de la logique des fonctions consommateurs. Pour en savoir plus sur la configuration du filtrage des événements et la définition des critères de filtre, consultez la section Filtrage des événements Lambda.

Réglage des AWS consommateurs Lambda

Les coûts peuvent être optimisés davantage en ajustant les paramètres de configuration Lambda, par exemple en augmentant la valeur `BatchSize` afin de traiter plus d'éléments par appel, en activant `BisectBatchOnFunctionError` pour empêcher le traitement des doublons (ce qui entraîne des coûts supplémentaires) et en définissant `MaximumRetryAttempts` pour limiter le nombre de nouvelles tentatives. Par défaut, les appels Lambda consommateurs ayant échoué font l'objet de nouvelles tentatives indéfiniment jusqu'à ce que l'enregistrement expire dans le flux, ce qui correspond à une durée d'environ 24 heures pour DynamoDB Streams et peut être compris entre 24 heures et un an pour Kinesis Data Streams, selon la configuration. Vous trouverez les options de configuration Lambda supplémentaires, y compris celles mentionnées ci-dessus pour les consommateurs du flux DynamoDB, dans le Guide du développeur AWS Lambda.

Évaluation de votre capacité provisionnée pour un dimensionnement approprié dans votre table DynamoDB

Cette section explique comment évaluer si vous avez alloué la capacité appropriée pour vos tables DynamoDB. À mesure que votre charge de travail évolue, vous devez modifier vos procédures opérationnelles de manière appropriée, notamment lorsque votre table DynamoDB est configurée en mode provisionné et que vous risquez de surprovisionner ou de sous-provisionner vos tables.

Les procédures décrites ci-dessous nécessitent des informations statistiques qui doivent être capturées à partir des tables DynamoDB qui prennent en charge votre application de production. Pour comprendre le comportement de votre application, vous devez définir une période suffisamment longue pour tenir compte de la saisonnalité de ses données. Par exemple, si votre application repose sur des cycles hebdomadaires, spécifier une période de trois semaines devrait vous laisser suffisamment de marge pour analyser ses besoins en matière de débit.

Si vous ne savez pas par où commencer, utilisez au moins un mois de données pour les calculs ci-dessous.

Lors de l'évaluation de la capacité, les tables DynamoDB peuvent configurer les unités de capacité de lecture RCUs () et les unités de capacité d'écriture (WCU) indépendamment. Si des index secondaires globaux (GSI) sont configurés sur vos tables, vous devez spécifier le débit qu'elles consommeront, qui sera également indépendant de la table de base RCUs et WCUs de celle-ci.

> **Note**
>
> Les index secondaires locaux (LSI) consomment la capacité à partir de la table de base.

Rubriques

- Comment récupérer les métriques de consommation sur vos tables DynamoDB
- Comment identifier les tables DynamoDB sous-provisionnées
- Comment identifier les tables DynamoDB surprovisionnées

Comment récupérer les métriques de consommation sur vos tables DynamoDB

Pour évaluer la table et la capacité du GSI, surveillez les CloudWatch mesures suivantes et sélectionnez la dimension appropriée pour récupérer les informations du tableau ou du GSI :

Unités de capacité de lecture	Unités de capacité d’écriture
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

Vous pouvez le faire via le AWS CLI ou le AWS Management Console.

AWS CLI

Avant de récupérer les indicateurs de consommation du tableau, nous devons commencer par capturer certains points de données historiques à l'aide de l' CloudWatch API.

Commencez par créer deux fichiers : `write-calc.json` et `read-calc.json`. Ces fichiers représentent les calculs pour une table ou un index secondaire global. Vous devrez mettre à jour certains champs, comme indiqué dans le tableau ci-dessous, pour les adapter à votre environnement.

Nom de champ	Définition	Exemple
`<table-name>`	Nom de la table que vous allez analyser	SampleTable
`<period>`	Période que vous utiliserez pour évaluer la cible d'utilisation, en secondes	Pour une période d'une heure, vous devez spécifier : 3 600
`<start-time>`	Le début de votre intervalle d'évaluation, spécifié au format ISO86 01	2022-02-21T23:00:00
`<end-time>`	La fin de votre intervalle d'évaluation, spécifiée au format ISO86 01	2022-02-22T06:00:00

Le fichier de calculs d'écriture récupérera le nombre de WCU provisionnées et consommées au cours de la période correspondant à la plage de dates spécifiée. Il générera également un pourcentage d'utilisation qui sera utilisé pour l'analyse. Le contenu entier du fichier `write-calc.json` doit se présenter comme suit :

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
```

```
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
```

```
  "MaxDatapoints": 24
}
```

Le fichier de calculs de lecture utilise un fichier similaire. Ce fichier récupérera combien RCUs ont été approvisionnés et consommés pendant la période correspondant à la plage de dates spécifiée. Le contenu du fichier `read-calc.json` doit se présenter comme suit :

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
```

```
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

Une fois les fichiers créés, vous pouvez commencer à récupérer les données d'utilisation.

1. Pour récupérer les données d'utilisation des écritures, émettez la commande suivante :

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Pour récupérer les données d'utilisation des lectures, émettez la commande suivante :

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

Le résultat des deux requêtes correspond à une série de points de données au format JSON qui serviront à l'analyse. Votre résultat dépendra du nombre de points de données que vous avez spécifiés, de la période et de vos données de charge de travail spécifiques. Voici ce à quoi il pourrait ressembler :

```
{
    "MetricDataResults": [
        {
```

```
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

> **Note**
>
> Si vous spécifiez une période courte et une plage de temps longue, vous devrez peut-être modifier la valeur `MaxDatapoints` par défaut, qui est de 24, dans le script. Cela représente un seul point de données par heure et donc 24 points de données par jour.

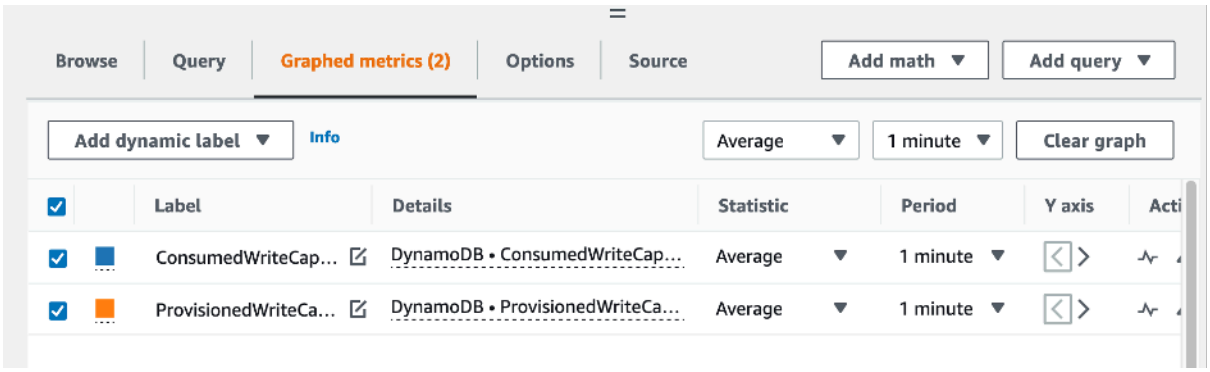
AWS Management Console

1. Connectez-vous AWS Management Console et accédez à la page CloudWatch de service. Sélectionnez une option appropriée Région AWS si nécessaire.
2. Dans la barre de navigation de gauche, sélectionnez la section Métriques, puis sélectionnez Toutes les métriques.

3. Cela ouvrira un tableau de bord avec deux panneaux. Le panneau supérieur affiche le graphique, tandis que le panneau inférieur contient les métriques que vous souhaitez représenter graphiquement. Choisissez DynamoDB.
4. Sélectionnez Métriques de table. Cette action vous montre les tables de votre région actuelle.
5. Utilisez le champ de recherche pour rechercher le nom de votre table et choisir les métriques de l'opération d'écriture : `ConsumedWriteCapacityUnits` et `ProvisionedWriteCapacityUnits`.

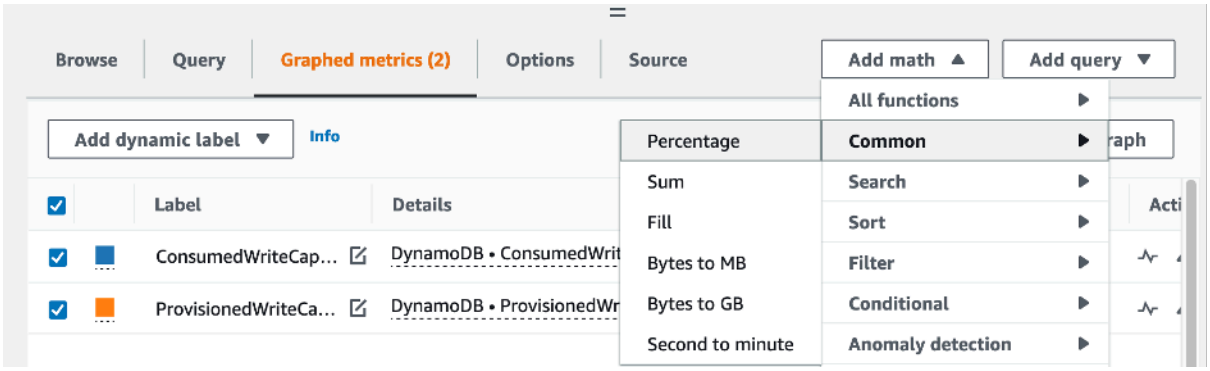
> **Note**
>
> Cet exemple décrit les métriques des opérations d'écriture, mais vous pouvez également suivre ces étapes pour représenter graphiquement les métriques des opérations de lecture.

6. Sélectionnez l'onglet Métriques graphiques (2) pour modifier les formules. Par défaut, CloudWatch sélectionne la fonction statistique Average pour les graphiques.

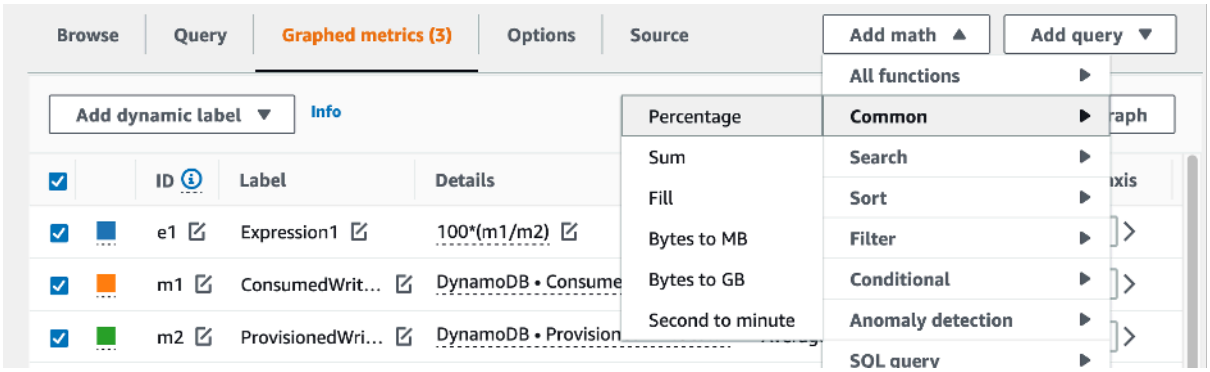


7. Lorsque les deux mesures représentées dans le graphique sont sélectionnées (case à cocher sur la gauche), sélectionnez le menu Add math (Ajouter une formule), suivi de Common (Commun), puis sélectionnez la fonction Percentage (Pourcentage). Répétez cette procédure deux fois.

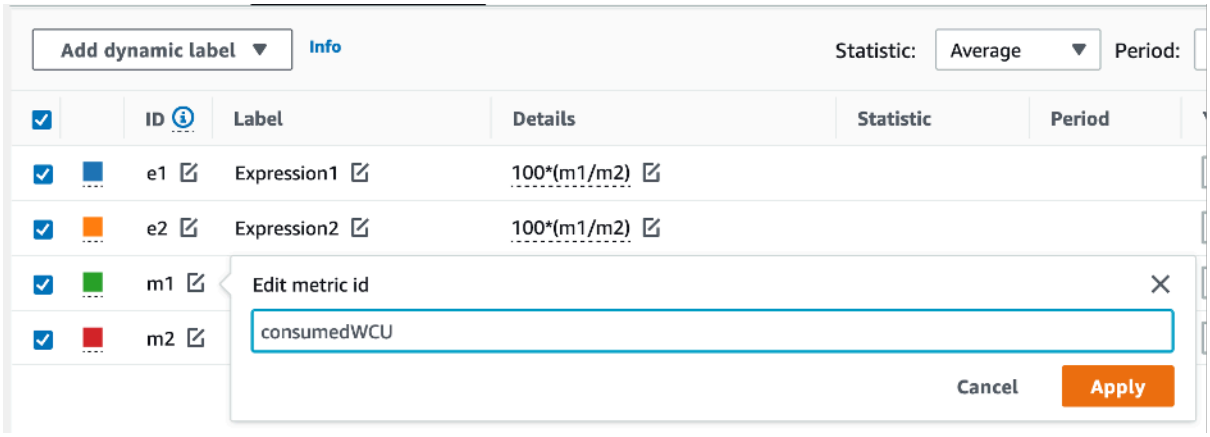
   Première sélection de la fonction Percentage (Pourcentage) :



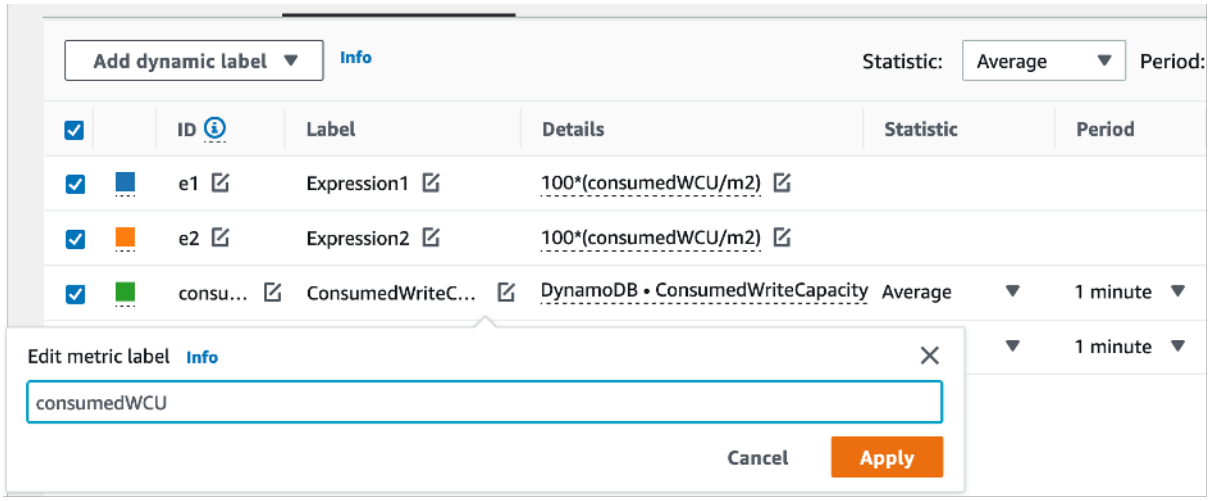
Deuxième sélection de la fonction Percentage (Pourcentage) :



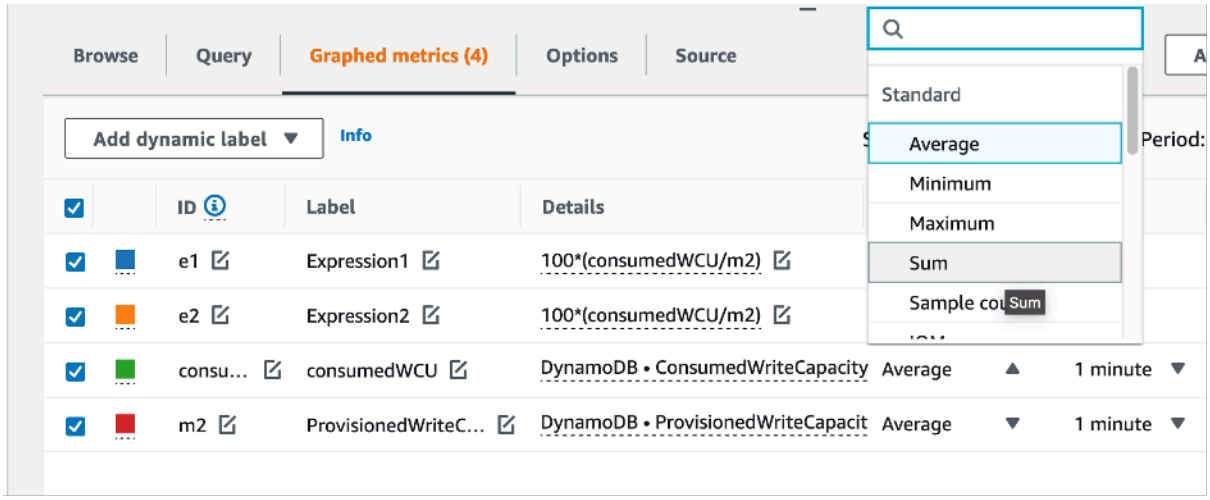
8. À ce stade, vous devriez avoir quatre métriques dans le menu inférieur. Travaillons sur le calcul de `ConsumedWriteCapacityUnits`. Par souci de cohérence, nous devons faire correspondre les noms de ceux que nous avons utilisés dans la AWS CLI section. Cliquez sur l'ID m1 et remplacez cette valeur par consumedWCU.

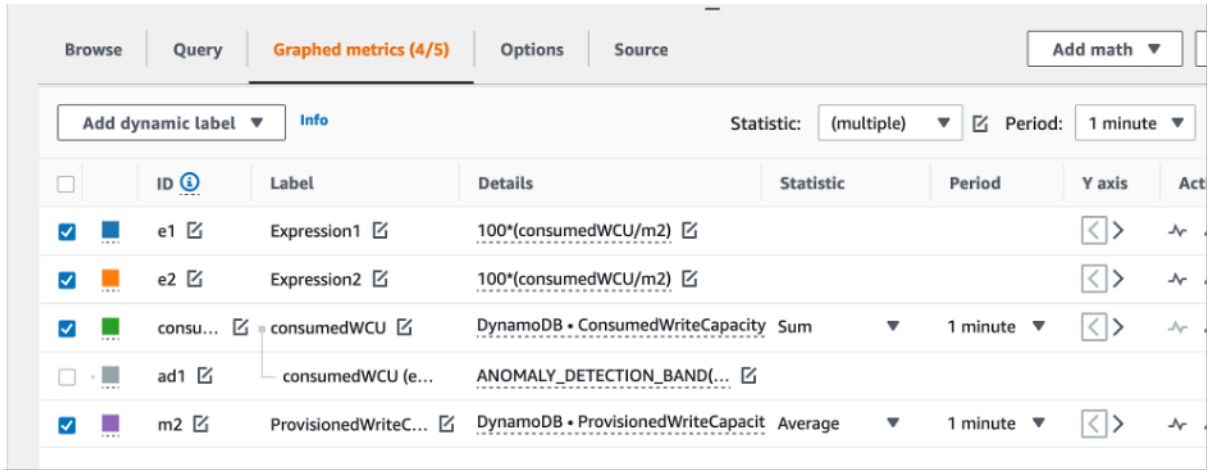


Renommez l'ConsumedWriteCapacityUnitétiquette en**`consumedWCU`**.

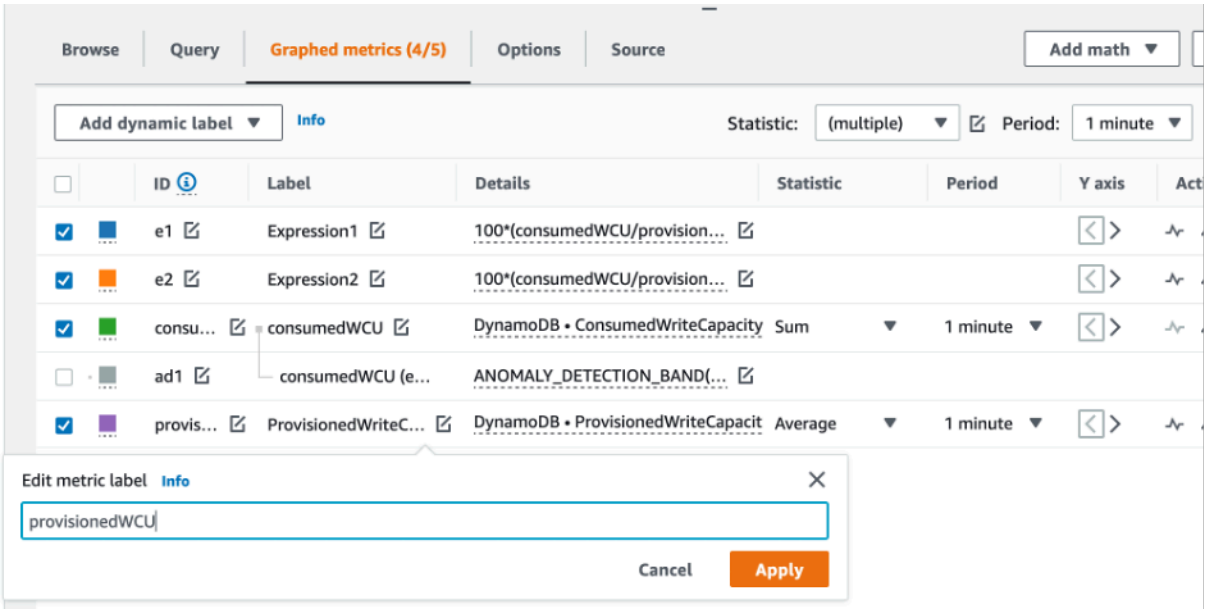


9. Remplacez la statistique Average (Moyenne) par Sum (Somme). Cette action crée automatiquement une autre métrique appelée ANOMALY_DETECTION_BAND. Dans le cadre de cette procédure, nous pouvons l'ignorer en supprimant la case à cocher au niveau de la métrique ad1 qui vient d'être générée.

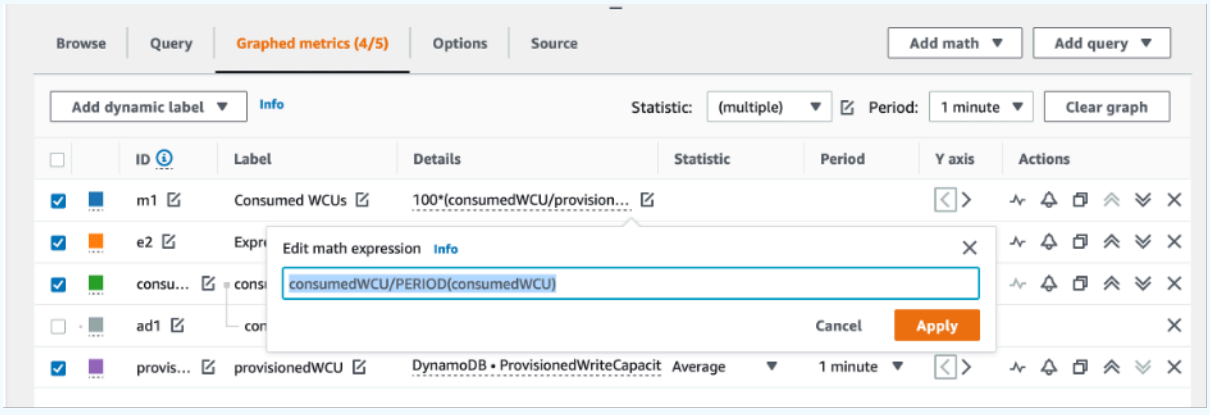




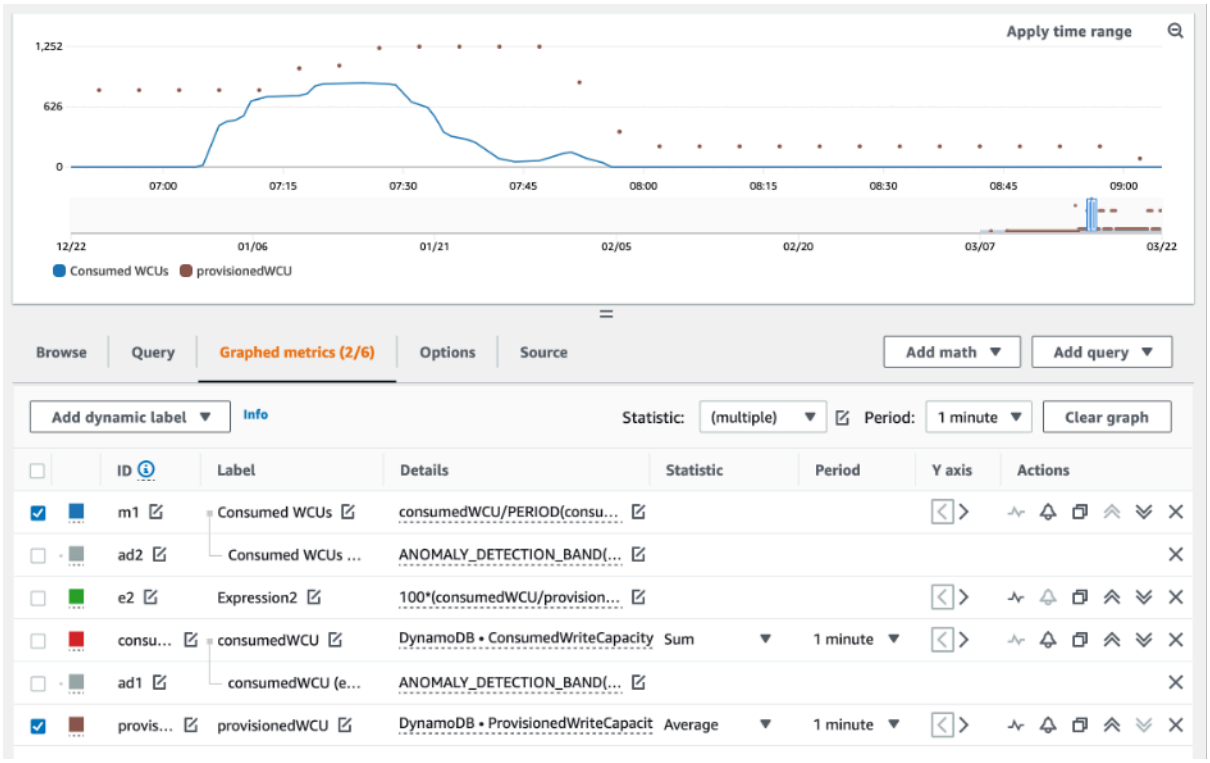
10. Répétez l'étape 8 pour remplacer le nom de l'ID m2 par provisionedWCU. Conservez la statistique Average (Moyenne).



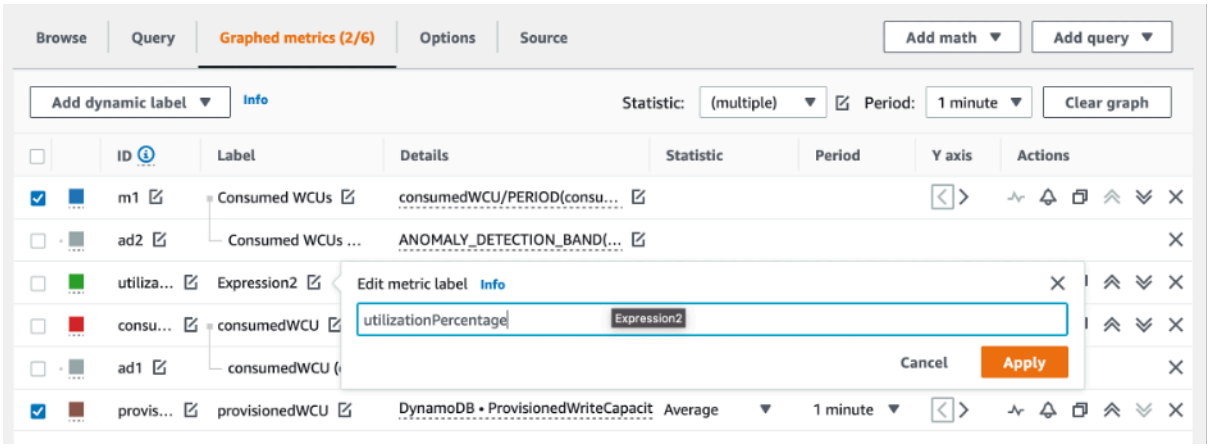
11. Sélectionnez l'étiquette Expression1 et mettez à jour la valeur en m1 et l'étiquette en Consumed. WCUs

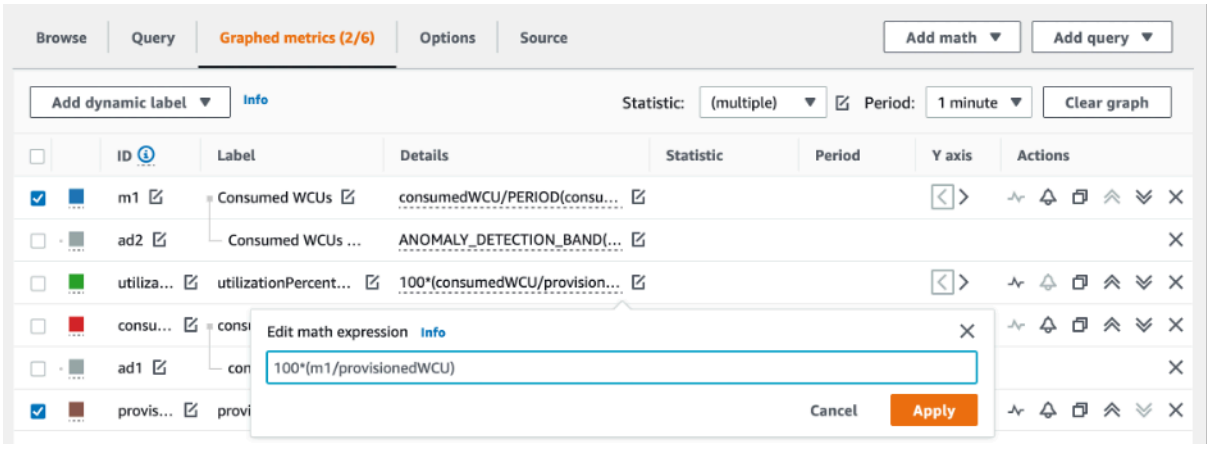
> **Note**
>
> Assurez-vous de n'avoir sélectionné que m1 (case à cocher sur la gauche) et provisionedWCU pour visualiser correctement les données. Mettez à jour la formule en cliquant sur Details (Détails) et en remplaçant la formule par consumedWCU/PERIOD(consumedWCU). Cette étape peut également générer une autre métrique ANOMALY_DETECTION_BAND, mais dans le cadre de cette procédure, nous pouvons l'ignorer.
>
> 
>

12. Vous devriez maintenant avoir deux graphiques : l'un qui indique votre provisionnement WCUs sur le tableau et l'autre qui indique la consommation WCUs. La forme du graphique peut être différente de celle ci-dessous, mais vous pouvez l'utiliser comme référence :



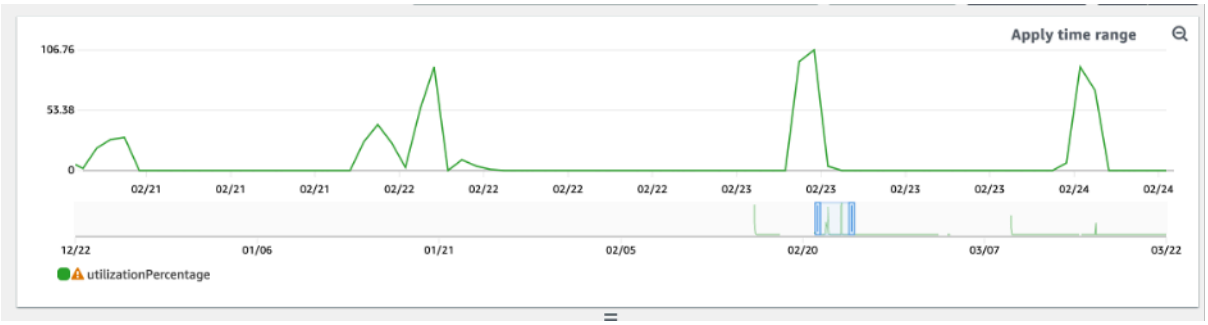
13. Mettez à jour la formule du pourcentage en sélectionnant le graphique Expression2 (e2). Renommez les étiquettes en IDs UtilizationPercentage. Remplacez le nom de la formule par 100*(m1/provisionedWCU).





14. Décochez la case de toutes les métriques, à l'exception de la métrique utilizationPercent pour visualiser vos modèles d'utilisation. L'intervalle par défaut est défini sur 1 minute, mais n'hésitez pas à le modifier selon vos besoins.

Voici une vue d'une période plus longue ainsi que d'une période d'une heure. Vous pouvez constater que l'utilisation était supérieure à 100 % à certains intervalles, mais cette charge de travail particulière présente des intervalles plus longs avec une utilisation nulle.



À ce stade, les résultats peuvent différer de ceux des images de cet exemple. Tout dépend des données de votre charge de travail. Les intervalles avec une utilisation supérieure à 100 % sont sujets à des événements de limitation. DynamoDB offre une capacité de débordement, mais dès que cette capacité est atteinte, tout ce qui dépasse 100 % est limité.

Comment identifier les tables DynamoDB sous-provisionnées

Pour la plupart des charges de travail, une table est considérée comme sous-provisionnée lorsqu'elle consomme constamment plus de 80 % de sa capacité provisionnée.

La capacité en rafale est une fonctionnalité de DynamoDB qui permet aux clients de consommer temporairement RCUs/WCUs plus que ce qui était initialement prévu (plus que le débit provisionné par seconde défini dans le tableau). La capacité de débordement a été créée pour absorber les augmentations soudaines du trafic dues à des événements spéciaux ou à des pics d'utilisation. Elle ne dure pas éternellement. Dès que les capacités inutilisées RCUs et épuisées WCUs sont épuisées, vous serez limité si vous essayez de consommer plus de capacité que celle prévue. Lorsque le trafic de votre application approche le taux d'utilisation de 80 %, le risque de limitation est nettement plus élevé.

La règle du taux d'utilisation de 80 % varie en fonction de la saisonnalité de vos données et de la croissance du trafic. Réfléchissez aux scénarios suivants :

- Si le trafic est resté stable à un taux d'utilisation d'environ 90 % au cours des 12 derniers mois, votre table dispose de la capacité idéale
- Si le trafic de vos applications augmente à un rythme de 8 % par mois en moins de 3 mois, vous allez atteindre une utilisation de 100 %
- Si le trafic de vos applications augmente à un rythme de 5 % en un peu plus de 4 mois, vous allez tout de même atteindre une utilisation de 100 %

Les résultats des requêtes ci-dessus donnent une idée de votre taux d'utilisation. Utilisez-les comme guide pour évaluer plus en détail d'autres métriques qui pourront vous aider à choisir d'augmenter la capacité de votre table selon vos besoins (par exemple, à un taux de croissance mensuel ou hebdomadaire). Travaillez avec l'équipe des opérations pour définir le pourcentage approprié pour votre charge de travail et vos tables.

Il existe des scénarios particuliers dans lesquels les données sont biaisées lorsque nous les analysons sur une base quotidienne ou hebdomadaire. Par exemple, pour les applications saisonnières qui connaissent des pics d'utilisation pendant les heures de travail (mais qui tombent ensuite presque à zéro en dehors de ces plages horaires), vous pourriez bénéficier de l'autoscaling planifié qui permet de spécifier les heures de la journée (et les jours de la semaine) pour lesquels vous souhaitez augmenter la capacité provisionnée, et quand la réduire. Au lieu de viser une capacité plus élevée pour couvrir les heures de pointe, vous pouvez également tirer parti des configurations de l'autoscaling des tables DynamoDB si la saisonnalité de vos données est moins prononcée.

> **Note**
>
> Lorsque vous créez une configuration de mise à l'échelle automatique de DynamoDB pour votre table de base, n'oubliez pas d'inclure une autre configuration pour les index secondaires globaux qui y sont associés.

Comment identifier les tables DynamoDB surprovisionnées

Les résultats de requête obtenus à partir des scripts ci-dessus fournissent les points de données nécessaires pour effectuer une analyse initiale. Si votre ensemble de données présente des valeurs d'utilisation inférieures à 20 % pendant plusieurs intervalles, votre table est peut-être surprovisionnée. Pour définir plus précisément s'il est nécessaire de réduire le nombre de RCUS WCUs et de RCUS, vous devez revoir les autres relevés dans les intervalles.

Lorsque des tables contiennent plusieurs intervalles d'utilisation faibles, vous pouvez tirer parti de l'utilisation de politiques d'autoscaling automatique en planifiant l'autoscaling automatique ou en configurant simplement les politiques d'autoscaling par défaut pour la table en fonction de l'utilisation.

Si votre charge de travail présente un faible ratio d'utilisation/accélération élevé (Max (ThrottleEvents) /Min (ThrottleEvents) dans l'intervalle), cela peut se produire lorsque vous avez une charge de travail très élevée où le trafic augmente considérablement pendant certains jours (ou heures), mais en général, le trafic est constamment faible. Dans ces scénarios, il peut être utile de recourir à l'autoscaling planifié.

Le AWSWell-Architected Framework aide les architectes du cloud à créer une infrastructure sécurisée, performante, résiliente et efficace pour une variété d'applications et de charges de travail. Construit autour de six piliers (excellence opérationnelle, sécurité, fiabilité, efficacité des performances, optimisation des coûts et durabilité),AWS Well-Architected propose une approche cohérente aux clients et aux partenaires pour évaluer les architectures et mettre en œuvre des conceptions évolutives.

Les objectifs AWSWell-Architected étendent les conseils proposés par AWS Well-Architected à des domaines industriels et technologiques spécifiques. Le cadre Well-Architected d'Amazon DynamoDB se concentre sur les charges de travail DynamoDB. Il fournit les bonnes pratiques, les principes de conception et les questions permettant d'évaluer et d'examiner une charge de travail DynamoDB. En procédant à une évaluation du cadre Well-Architected d'Amazon DynamoDB, vous recevrez des informations et des conseils sur les principes de conception recommandés en lien avec chacun

des piliers AWS Well-Architected. Ces conseils sont basés sur notre expérience de travail avec des clients de différents secteurs, segments, tailles et zones géographiques.

À l'issue de l'évaluation du cadre Well-Architected, vous recevrez un résumé de recommandations pratiques afin d'optimiser et d'améliorer votre charge de travail DynamoDB.

Réalisation de l'évaluation du cadre Well-Architected d'Amazon DynamoDB

La révision DynamoDB Well-Architected Lens est généralement réalisée AWS par un architecte de solutions en collaboration avec le client, mais elle peut également être réalisée par le client en libre-service. Nous vous recommandons d'évaluer les six piliers Well-Architected dans le contexte du cadre Well-Architected d'Amazon DynamoDB, mais vous pouvez également décider de commencer par concentrer votre attention sur un ou plusieurs piliers.

Des informations et instructions supplémentaires pour effectuer un examen d'Amazon DynamoDB Well-Architected Lens sont disponibles dans cette vidéo et sur la page DynamoDB Well-Architected Lens. GitHub

Piliers du cadre Well-Architected d'Amazon DynamoDB

Le cadre Well-Architected d'Amazon DynamoDB repose sur six piliers :

Pilier d'efficacité des performances

Le pilier d'efficacité des performances englobe la capacité à utiliser efficacement les ressources informatiques pour satisfaire aux exigences système et à maintenir cette efficacité au fur et à mesure que la demande change et que les technologies évoluent.

Les principes majeurs de conception de DynamoDB pour ce pilier concernent la modélisation des données, le choix des clés de partition et des clés de tri, et la définition des index secondaires en fonction des schémas d'accès aux applications. Les autres considérations incluent le choix du mode de débit optimal pour la charge de travail, le réglage du AWS SDK et, le cas échéant, l'utilisation d'une stratégie de mise en cache optimale. Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier d'efficacité des performances du cadre Well-Architected de DynamoDB.

Pilier d'optimisation des coûts

Le pilier d'optimisation des coûts vise à éviter les coûts inutiles.

Les sujets clés sont les suivants : compréhension et contrôle des dépenses, sélection du nombre de types de ressources le plus approprié et le plus juste, analyse des dépenses dans le temps, conception de vos modèles de données afin d'optimiser le coût des schémas d'accès spécifiques aux applications, et mise à l'échelle pour répondre aux besoins de l'entreprise sans engager de dépenses excessives.

Les principes clés d'optimisation des coûts pour DynamoDB consistent à choisir le mode de capacité et la classe de table les plus appropriés pour vos tables et à éviter le surdimensionnement de la capacité en utilisant le mode de capacité à la demande ou le mode de capacité allouée avec mise à l'échelle automatique. Parmi les autres considérations, citons la modélisation et l'interrogation efficaces des données afin de réduire la quantité de capacité consommée, la réservation de portions de la capacité consommée à prix réduit, la réduction de la taille des éléments, l'identification et la suppression des ressources non utilisées et l'utilisation de la durée de vie pour supprimer automatiquement et gratuitement les données périmées. Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier d'optimisation des coûts du cadre Well-Architected de DynamoDB.

Consultez la section Optimisation des coûts pour plus d'informations sur les bonnes pratiques en matière d'optimisation des coûts pour DynamoDB.

Pilier d'excellence opérationnelle

Le pilier d'excellence opérationnelle se concentre sur l'exécution et la surveillance des systèmes afin de générer de la valeur opérationnelle, ainsi que sur l'amélioration continue des processus et des procédures. Les sujets clés sont les suivants : automatisation des modifications, réponse aux événements et définition des normes pour gérer les opérations quotidiennes.

Les principaux principes de conception relatifs à l'excellence opérationnelle de DynamoDB incluent la surveillance des métriques DynamoDB via CloudWatch AWS Config Amazon, ainsi que l'alerte et la correction automatiques lorsque des seuils prédéfinis sont dépassés ou que des règles non conformes sont détectées. D'autres considérations concernent la définition des ressources DynamoDB par l'intermédiaire de l'infrastructure sous forme de code et l'exploitation des balises pour une meilleure organisation, une identification et une comptabilisation des coûts de vos ressources DynamoDB. Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier d'excellence opérationnelle du cadre Well-Architected de DynamoDB.

Pilier de fiabilité

Le pilier de fiabilité vise à garantir qu'une charge de travail exécute correctement et de manière cohérente la fonction prévue. Une charge de travail résiliente se rétablit rapidement après une

défaillance pour répondre à la demande des entreprises et des clients. Les sujets clés sont les suivants : conception des systèmes distribués, planification de la restauration et gestion des modifications.

Les principes essentiels de conception de fiabilité de DynamoDB consistent à choisir la stratégie de sauvegarde et de rétention en fonction de vos exigences en matière de RPO et de RTO, à utiliser des tables globales DynamoDB pour les charges de travail multirégionales ou des scénarios de reprise après sinistre entre régions avec un faible RTO, à implémenter une logique de nouvelle tentative avec un retard exponentiel dans l'application en configurant et en utilisant ces fonctionnalités dans le SDK AWS, et à surveiller les métriques DynamoDB via Amazon et à envoyer automatiquement des alertes et correction lorsque des seuils prédéfinis sont dépassés. CloudWatch Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier de fiabilité du cadre Well-Architected de DynamoDB.

Pilier de sécurité

Le pilier de sécurité se concentre sur la protection des informations et des systèmes. Les sujets clés sont les suivants : confidentialité et intégrité des données, identification et gestion de qui peut faire quoi grâce à la gestion des privilèges, protection des systèmes et mise en place de contrôles pour détecter les événements de sécurité.

Les principes majeurs de conception de la sécurité de DynamoDB sont le chiffrement des données en transit avec HTTPS, le choix du type de clés pour le chiffrement des données au repos et la définition des rôles et des politiques IAM pour authentifier, autoriser et fournir un accès précis aux ressources DynamoDB. Les autres considérations incluent l'audit du plan de contrôle DynamoDB et des opérations du plan de données via.AWS CloudTrail Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier de sécurité du cadre Well-Architected de DynamoDB.

Consultez la section Sécurité pour plus d’informations sur la sécurité de DynamoDB.

Pilier de durabilité

Le pilier de durabilité vise à minimiser les impacts environnementaux liés à l’exécution de charges de travail dans le cloud. Les sujets clés sont les suivants : modèle de responsabilité partagée pour la durabilité, compréhension de l’impact et maximisation de l’utilisation afin de minimiser les ressources requises et de réduire les impacts en aval.

Les principes majeurs de conception de la durabilité de DynamoDB comprennent l’identification et la suppression des ressources DynamoDB non utilisées, la prévention du surprovisionnement par

l'utilisation du mode de capacité à la demande ou du mode de capacité allouée avec mise à l'échelle automatique, l'interrogation efficace pour réduire la quantité de capacité consommée et la réduction de l'empreinte du stockage par la compression des données et la suppression des données périmées grâce à l'utilisation de la durée de vie. Pour en savoir plus sur ces principes de conception, regardez cette vidéo détaillée sur le pilier de durabilité du cadre Well-Architected de DynamoDB.

Bonnes pratiques pour la conception et l'utilisation performantes de clés de partition dans DynamoDB

La clé primaire qui identifie de façon unique chaque élément dans un table Amazon DynamoDB peut être simple (uniquement une clé de partition) ou composite (une clé de partition associée à une clé de tri).

Vous devez concevoir votre application afin d'obtenir une activité uniforme sur toutes les clés de partition de la table et de ses index secondaires. Vous pouvez déterminer les modèles d'accès dont votre application a besoin et estimer les unités de capacité d'écriture et de lecture dont chaque table et index secondaire a besoin.

> **Note**
>
> La capacité adaptative s'applique au mode à la demande et à la capacité provisionnée.

Chaque partition d'une table DynamoDB est conçue pour fournir une capacité maximale de 3 000 unités de lecture par seconde et de 1 000 unités d'écriture par seconde. Une unité de lecture équivaut à une opération de lecture fortement cohérente par seconde, ou à deux opérations de lecture cohérente à terme par seconde, pour un élément dont la taille peut atteindre 4 Ko. Une unité d'écriture équivaut à une opération d'écriture par seconde, pour un élément d'une taille pouvant atteindre 1 Ko.

Vous devez tenir compte de la taille de l'élément lorsque vous évaluez les limites de débit de partition pour votre table. Par exemple, si la taille des éléments de la table est de 20 Ko, une seule opération de lecture cohérente consommera 5 unités de lecture. Cela signifie que vous pouvez effectuer simultanément 600 opérations de lecture cohérentes par seconde sur cet élément unique avant d'atteindre les limites de partition. Le débit total sur toutes les partitions de la table peut être limité par le débit provisionné en mode provisionné ou par la limite de débit au niveau de la table en mode à la demande. Pour de plus amples informations, veuillez consulter Service Quotas.

Rubriques

- Conception des clés de partition de manière à répartir votre charge de travail dans DynamoDB
- Utilisation du partitionnement d'écriture pour une répartition équitable des charges de travail dans votre table DynamoDB
- Distribution efficace de l'activité d'écriture pendant le chargement de données dans DynamoDB

Conception des clés de partition de manière à répartir votre charge de travail dans DynamoDB

La partie clé de partition de la clé primaire d'une table détermine les partitions logiques dans lesquelles les données d'une table sont stockées. Cela a ensuite un impact sur les partitions physiques sous-jacentes. Une conception de clé de partition qui ne distribue pas les I/O demandes de manière efficace peut créer des partitions « chaudes » qui se traduisent par une limitation et une utilisation inefficace de la capacité allouée. I/O

L'utilisation optimale du débit alloué d'une table ne dépend pas seulement des modèles de charge de travail des éléments individuels, mais également de la conception de la clé de partition. Cela ne signifie pas que vous devez accéder à toutes les valeurs de clé de partition pour atteindre un niveau de débit efficace, ni même que le pourcentage de valeurs de clé de partition accédées doit être élevé. Cela signifie que plus votre charge de travail accède à des valeurs de clé de partition distinctes, plus ces demandes sont réparties dans l'espace partitionné. En général, vous utilisez le débit qui vous est alloué plus efficacement lorsque le rapport entre les valeurs de clé de partition accédées et le nombre total de valeurs de clé de partition d'une table augmente.

Voici une comparaison de l'efficacité du débit alloué de plusieurs schémas courants de clé de partition.

Valeur de la clé de partition	Uniformité
ID utilisateur, quand l'application a de nombreux utilisateurs.	Bon
Code d'état, où il existe seulement quelques codes d'état possibles.	Mauvaise

Valeur de la clé de partition	Uniformité
Date de création de l'élément, arrondie à la période la plus proche (par exemple, jour, heure ou minute).	Mauvaise
ID d'appareil, où chaque appareil accède aux données à des intervalles relativement similaires.	Bon
ID d'appareil où, même si un grand nombre d'appareils est suivi, l'un est bien plus populaire que tous les autres.	Mauvaise

Si une seule table n'a qu'un petit nombre de valeurs de clé de partition, pensez à répartir vos opérations d'écriture entre des valeurs de clé de partition plus distinctes. En d'autres termes, structurez les éléments de clé primaire pour éviter une valeur de partition « critique » (fortement demandée) qui ralentit les performances globales.

Par exemple, imaginons une table avec une clé primaire composite. La clé de partition représente la date de création de l'élément, arrondie au jour le plus proche. La clé de tri est un identificateur d'élément. Sur un jour donné, disons le `2014-07-09`, tous les nouveaux éléments sont écrits sur cette même valeur de clé de partition (et sur la partition physique correspondante).

Si la table s'adapte entièrement à une seule partition (en tenant compte de la croissance de vos données au fil du temps) et si les exigences en débit de lecture et d'écriture de votre application ne dépassent pas les capacités en lecture et en écriture d'une seule partition, votre application ne doit rencontrer aucune limitation inattendue comme résultat du partitionnement.

Pour utiliser NoSQL Workbench pour DynamoDB afin de mieux visualiser votre conception de clé de partition, consultez Création de modèles de données avec NoSQL Workbench.

Utilisation du partitionnement d'écriture pour une répartition équitable des charges de travail dans votre table DynamoDB

Une manière de mieux distribuer les écritures sur un espace de clé de partition dans Amazon DynamoDB consiste à développer l'espace. Vous pouvez effectuer cette opération de plusieurs manières. Vous pouvez ajouter un nombre aléatoire aux valeurs de clé de partition pour répartir les

éléments entre les partitions. Ou vous pouvez utiliser un nombre qui est calculé en fonction d'une information sur laquelle porte la requête.

Partitionnement à l'aide de suffixes aléatoires

L'ajout d'un nombre aléatoire à la fin des valeurs de clé de partition constitue une stratégie permettant de répartir plus équitablement des charges sur un espace de clé de partition. Vous pouvez alors randomiser les écritures sur l'espace plus important.

Par exemple, dans le cas d'une clé de partition représentant la date du jour, vous pouvez choisir un nombre aléatoire compris entre `1` et `200` et le concaténer en tant que suffixe avec la date. Cela génère des valeurs de clé de partition telles que `2014-07-09.1`, `2014-07-09.2`, et ainsi de suite jusqu'à `2014-07-09.200`. Puisque vous randomisez la clé de partition, les écritures quotidiennes sur la table sont réparties uniformément entre plusieurs partitions. Cela permet un meilleur parallélisme et un débit général plus élevé.

Toutefois, pour lire tous les éléments d'un jour donné, vous devrez interroger les éléments et rechercher tous les suffixes, puis fusionner les résultats. Par exemple, vous devez d'abord lancer une requête `Query` pour la valeur de clé de partition `2014-07-09.1`. Ensuite, émettez une autre `Query` pour `2014-07-09.2`, et ainsi de suite, jusqu'à `2014-07-09.200`. Enfin, votre application doit fusionner les résultats de toutes ces demandes `Query`.

Partitionnement à l'aide de suffixes calculés

Une stratégie de randomisation peut considérablement améliorer le débit d'écriture. Cependant, il est difficile de lire un élément spécifique, car vous ne connaissez pas la valeur de suffixe utilisée lors de l'écriture de l'élément. Pour faciliter la lecture d'éléments individuels, vous pouvez utiliser une autre stratégie. Plutôt que d'utiliser un nombre aléatoire pour répartir les éléments sur les partitions, utilisez un nombre que vous pouvez calculer selon l'objet sur lequel vous souhaitez faire porter l'interrogation.

Considérons l'exemple précédent, dans lequel une table utilise la date du jour dans la clé de partition. Supposons maintenant que chaque élément a un attribut `OrderId` accessible, et que vous devez le plus souvent rechercher des éléments par ID de commande en plus de la date. Avant que votre application n'écrive l'élément sur la table, elle peut calculer un suffixe de hachage en fonction de l'ID de commande et l'ajouter à la date de clé de partition. Le calcul peut se traduire par un nombre compris entre 1 et 200 assez uniformément distribué, à l'image de ce que la stratégie de randomisation produit.

Un simple calcul suffirait probablement, tel que le produit des valeurs de point de code UTF-8 pour les caractères de l'ID de commande, modulo 200, + 1. La valeur de clé de partition sera alors la date concaténée avec le résultat du calcul.

Avec cette stratégie, les écritures sont réparties uniformément entre les valeurs de clé de partition, et de ce fait entre les partitions physiques. Vous pouvez facilement exécuter une opération `GetItem` pour un élément et une date donnés, car vous pouvez calculer la valeur de clé de partition pour une valeur `OrderId` spécifique.

Pour lire tous les éléments pour un jour donné, vous devez toujours interroger (`Query`) chacune des clés `2014-07-09.N` (où `N` est 1–200), et votre application doit ensuite fusionner tous les résultats. L'avantage est que vous évitez qu'une valeur de clé de partition « critique » ne prenne l'ensemble de la charge de travail.

> **Note**
>
> Pour une stratégie encore plus efficace conçue spécialement pour gérer des données en séries chronologiques volumineuses, consultez Données de séries temporelles.

Distribution efficace de l'activité d'écriture pendant le chargement de données dans DynamoDB

En règle générale, lorsque vous chargez des données d'autres sources, Amazon DynamoDB partitionne vos données de table sur plusieurs serveurs. Vous obtenez de meilleures performances en chargeant des données sur tous les serveurs alloués simultanément.

Par exemple, supposons que vous souhaitiez charger des messages utilisateur dans une table DynamoDB utilisant une clé primaire composite avec `UserID` comme clé de partition et `MessageID` comme clé de tri.

Lorsque vous chargez les données, une approche que vous pouvez adopter consiste à charger tous les éléments de message pour chaque utilisateur, un utilisateur après l'autre :

UserID	MessageId
U1	1
U1	2

UserID	MessageId
U1	...
U1	... jusqu’à 100
U2	1
U2	2
U2	...
U2	... jusqu’à 200

Le problème dans ce cas est que vous ne distribuez pas vos demandes d’écriture à DynamoDB dans vos valeurs de clé de partition. Vous prenez une valeur de clé de partition à la fois et chargez tous ses éléments avant de passer à la valeur de clé de partition suivante et d’effectuer la même opération.

En coulisses, DynamoDB partitionne les données dans votre table sur plusieurs serveurs. Pour utiliser pleinement toute la capacité de débit approvisionnée pour la table, vous devez distribuer votre charge de travail entre les valeurs de clé de votre partition. En dirigeant une quantité inégale de travail de chargement vers des éléments ayant tous la même valeur de clé de partition, vous n’utilisez pas complètement toutes les ressources que DynamoDB a approvisionnées pour votre table.

Vous pouvez distribuer votre travail de chargement à l’aide de la clé de tri pour charger un élément de chaque valeur de clé de partition, puis un autre, et ainsi de suite :

UserID	MessageId
U1	1
U2	1
U3	1
...	...
U1	2

UserID	MessageId
U2	2
U3	2
...	...

Chaque chargement de cette séquence utilise une valeur de clé de partition différente, ce qui permet d'occuper simultanément plus de serveurs DynamoDB et d'améliorer vos performances de débit.

Bonnes pratiques concernant l'utilisation de clés de tri pour organiser les données dans DynamoDB

Dans une table Amazon DynamoDB, la clé primaire qui identifie de façon unique chaque élément peut se composer d'une clé de partition et d'une clé de tri.

Les clés de tri bien conçues présentent deux avantages :

- Elles rassemblent des informations connexes dans un seul emplacement, ce qui permet de les interroger efficacement. Une conception soignée de la clé de tri vous permet d'extraire les groupes d'éléments connexes les plus couramment demandés à l'aide de requêtes de plage utilisées avec les opérateurs tels que `begins_with`, `between`, >, <, etc.
- Les clés de tri composites vous permettent de définir des relations hiérarchiques (one-to-many) dans vos données que vous pouvez interroger à n'importe quel niveau de la hiérarchie.

  Par exemple, dans une table répertoriant des emplacements géographiques, vous pouvez structurer la clé de tri comme suit :

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  Cela vous permet de créer des requêtes de plage efficaces concernant la liste des emplacements à n'importe lequel de ces niveaux d'agrégation, de `country` jusqu'à `neighborhood`, en incluant tout ce qu'il y a entre les deux.

Utilisation de clés de tri pour le contrôle de version

De nombreuses applications ont besoin d'avoir un historique des révisions au niveau des éléments à des fins d'audit ou de conformité, et ont également besoin de pouvoir extraire facilement la version la plus récente. Il existe un modèle de conception performant qui permet d'atteindre cet objectif en utilisant des préfixes de clé de tri :

- Pour chaque nouvel élément, créez deux copies de l'élément : une copie doit comporter un préfixe de numéro de version égal à zéro (comme `v0_`) au début de la clé de tri, et l'autre doit comporter un préfixe de numéro de version égal à un (comme `v1_`).
- Chaque fois que l'élément est mis à jour, utilisez le préfixe de version supérieur suivant pour la clé de tri de la version mise à jour et copiez le contenu mis à jour dans l'élément dont le préfixe de version est zéro. Vous pouvez ainsi localiser facilement la dernière version de n'importe quel élément en utilisant le préfixe zéro.

Par exemple, un fabricant de pièces détachées peut utiliser un schéma comme celui illustré ci-après.

Primary Key		Data-Item Attributes...				
Partition Key *Equipment_ID*	Sort Key *(varies)*	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
Equipment_1	Details	Name: Biphasic Cardiometer *(equipment name)*	Factory_ID: S14_Tukwila *(factory where manufactured)*	Line_ID R_7 *(assembly-line ID)*		
	v0_Audit	Auditor: Padma *(name of the auditor)*	Latest: 3 *(most recent audit version)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	...etc.
	v1_Audit	Auditor: Rick *(name of the auditor)*	Time: 2018-03-14T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943922EKG14 *(detailed problem report in S3)*	...etc.
	v2_Audit	Auditor: George *(name of the auditor)*	Time: 2018-03-18T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943923EKG15 *(detailed problem report in S3)*	...etc.
	v3_Audit	Auditor: Padma *(name of the auditor)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	Report: x792 *(pass confirmation report)*	...etc.

L'élément `Equipment_1` passe par une série d'audits effectués par plusieurs auditeurs. Les résultats de chaque nouvel audit sont capturés dans un nouvel élément de la table, en commençant par le numéro de version un et en incrémentant ce numéro à chaque nouvelle version.

Chaque fois qu'une nouvelle révision est ajoutée, la couche d'application remplace le contenu de l'élément de version zéro (dont la clé de tri est égale à `v0_Audit`) par celui de la nouvelle révision.

Chaque fois que l'application doit extraire le statut d'audit le plus récent, elle peut interroger le préfixe de clé de tri `v0_`.

Si l'application a besoin d'extraire la totalité de l'historique des révisions, elle peut interroger tous les éléments figurant sous la clé de partition de l'élément et filtrer l'élément `v0_`.

Cette conception fonctionne également pour les audits portant sur plusieurs parties d'un équipement, si vous incluez la pièce individuelle IDs dans la clé de tri après le préfixe de la clé de tri.

Bonnes pratiques d'utilisation d'index secondaires dans DynamoDB

Les index secondaires sont souvent essentiels pour prendre en charge les modèles de requête requis par votre application. En même temps, une utilisation excessive ou inefficace des index secondaires peut inutilement augmenter les coûts et diminuer les performances.

Table des matières

- Consignes générales pour les index secondaires dans DynamoDB
  - Utiliser les index efficacement
  - Choisir soigneusement les projections
  - Optimiser les requêtes fréquentes pour éviter les extractions
  - Prendre en compte les limites de taille de la collection d'éléments lors de la création des index secondaires locaux
- Tirer profit des index partiellement alloués
  - Exemples d'index fragmentés dans DynamoDB
- Utilisation d'index secondaire globaux pour les requêtes de regroupement matérialisé dans DynamoDB
- Surcharge des index secondaires globaux dans DynamoDB
- Utilisation d'un partitionnement d'écriture d'index secondaire global pour des requêtes de table sélectives dans DynamoDB
  - Conception de modèle
  - Stratégie de partition
  - Interrogation du GSI partitionné
  - Considérations relatives à l'exécution parallèle des requêtes
  - Exemple de code
- Utilisation d'index secondaires globaux pour créer un réplica éventuellement cohérent dans DynamoDB

Consignes générales pour les index secondaires dans DynamoDB

Amazon DynamoDB prend en charge deux types d'index secondaires :

- Index secondaires globaux (GSI) : index dotés d'une clé de partition et d'une clé de tri qui peuvent être différentes de celles de la table de base. Un index secondaire global est considéré comme « global », car les requêtes sur l'index peuvent couvrir toutes les données de la table de base, sur toutes les partitions. Un index secondaire global n'a pas de limitations de taille et possède ses propres paramètres de débit alloué pour les activités de lecture et d'écriture qui sont distincts de ceux de la table.
- Index secondaires locaux : index dotés de la même clé de partition que celle de la table de base, mais d'une clé de tri différente. Un index secondaire est « local » dans la mesure où la portée de chacune de ses partitions correspond à une partition de la table de base ayant la même valeur de clé de partition. Par conséquent, la taille totale des éléments indexés pour chaque valeur de clé de partition ne doit pas dépasser 10 Go. En outre, un index secondaire local partage les paramètres de débit alloué pour les activités de lecture et d'écriture avec la table qu'il indexe.

Chaque table dans DynamoDB peut comporter jusqu'à 20 index secondaires globaux (quota par défaut) et 5 index secondaires locaux.

Les index secondaires globaux sont souvent plus utiles que les index secondaires locaux. La décision de choisir un type d'index plutôt que l'autre peut aussi être dictée par les exigences de votre application. Pour obtenir une comparaison des index secondaires globaux et des index secondaires locaux et pour savoir comment choisir entre les deux, consultez the section called “Utilisation des index”.

Voici quelques principes généraux et modèles de conception à garder à l'esprit lorsque vous créez des index dans DynamoDB :

Rubriques

- Utiliser les index efficacement
- Choisir soigneusement les projections
- Optimiser les requêtes fréquentes pour éviter les extractions
- Prendre en compte les limites de taille de la collection d'éléments lors de la création des index secondaires locaux

Utiliser les index efficacement

Réduisez au maximum le nombre d'index. Ne créez pas des index secondaires sur des attributs que vous n'interrogez pas souvent. Les index rarement utilisés contribuent à augmenter le stockage et les I/O coûts sans améliorer les performances des applications.

Choisir soigneusement les projections

Comme les index secondaires consomment du stockage et du débit alloué, vous devez maintenir la taille de l'index aussi réduite que possible. En outre, plus l'index est petit, meilleures sont les performances par rapport à l'interrogation de la table complète. Si vos requêtes ne renvoient généralement qu'un petit sous-ensemble d'attributs et que la taille totale de ces attributs est beaucoup plus petite que l'élément entier, projetez uniquement les attributs que vous demandez régulièrement.

Si vous prévoyez une importante activité d'écriture sur une table, par rapport aux lectures, respectez les bonnes pratiques suivantes :

- Pensez à projeter moins d'attributs pour réduire la taille des éléments écrits sur l'index. Cependant, ceci s'applique uniquement si la taille des attributs projetés serait sinon supérieure à une unité de d'écriture (1 Ko). Par exemple, si la taille d'une entrée d'index n'est que de 200 octets, DynamoDB l'arrondit à 1 Ko. En d'autres termes, aussi longtemps que les éléments de l'index sont petits, vous pouvez projeter plus d'attributs sans aucun coût supplémentaire.
- Évitez de projeter des attributs lorsque vous savez qu'ils sont rarement nécessaires dans des requêtes. Chaque fois que vous mettez à jour un attribut qui est projeté dans un index, les frais supplémentaires pour la mise à jour de l'index vous seront également facturés. Vous pouvez toujours récupérer les attributs non projetés dans une opération `Query` à un coût de débit alloué plus élevé, mais les coûts des requêtes peuvent être sensiblement plus bas que les coûts de mise à jour de l'index.
- Spécifiez `ALL` uniquement si vous voulez que vos requêtes retournent la totalité de l'élément de table trié sur une clé de tri différente. La projection de tous les attributs permet d'éliminer la nécessité des extractions de table, mais, dans la plupart des cas, cela double vos coûts de stockage et d'activité en écriture.

Trouvez un équilibre entre la nécessité de maintenir la taille de l'index aussi réduite que possible et celle de réduire au minimum les extractions, comme expliqué dans la section suivante.

Optimiser les requêtes fréquentes pour éviter les extractions

Pour obtenir les requêtes les plus rapides avec la plus faible latence possible, projetez tous les attributs que ces requêtes sont censées renvoyer. En particulier, si vous interrogez un index secondaire local pour des attributs non projetés, DynamoDB extrait automatiquement ces attributs de la table, ce qui nécessite de lire la totalité de l'élément à partir de la table. Cela introduit de la latence et I/O des opérations supplémentaires que vous pouvez éviter.

Gardez à l'esprit que les requêtes « occasionnelles » peuvent souvent se transformer en requêtes « essentielles ». S'il existe des attributs que vous ne prévoyez pas de projeter, car vous pensez ne les interroger que rarement, demandez-vous si les circonstances peuvent changer et si vous pourriez regretter ultérieurement de ne pas projeter ces attributs.

Pour plus d'informations sur les extractions de table, consultez Considérations relatives au débit alloué pour les index secondaires locaux.

Prendre en compte les limites de taille de la collection d'éléments lors de la création des index secondaires locaux

Une collection d'éléments regroupe tous les éléments d'une table et ses index secondaires locaux qui ont la même clé de partition. Aucune collection d'éléments ne peut pas dépasser 10 Go. Il se peut donc qu'une valeur de clé de partition particulière manque d'espace.

Lorsque vous ajoutez ou mettez à jour un élément de table, DynamoDB met à jour tous les index secondaires locaux concernés. Si les attributs indexés sont définis dans la table, la taille des index secondaires locaux augmente également.

Lorsque vous créez un index secondaire local, pensez à la quantité de données qui seront écrites dans celui-ci et à la quantité de ces éléments de données qui auront la même valeur de clé de partition. Si vous pensez que la somme des éléments de table et d'index d'une valeur de clé de partition particulière risque de dépasser 10 Go, interrogez-vous sur l'opportunité d'éviter de créer l'index.

Si vous ne pouvez pas éviter la création de l'index secondaire local, vous devez anticiper la taille limite de la collection d'éléments et prendre les mesures nécessaires avant qu'elle ne soit dépassée. Une bonne pratique consiste à utiliser le paramètre `ReturnItemCollectionMetrics` lorsque vous rédigez des éléments pour surveiller et signaler les tailles de collection d'éléments approchant la limite de 10 Go. Le dépassement de la taille maximale de la collection d'éléments entraînera l'échec des tentatives d'écriture. Vous pouvez atténuer les problèmes liés à la taille des collections

d'éléments en surveillant leur taille et en recevant des alertes avant qu'elles n'affectent votre demande.

> **Note**
>
> Une fois créé, un index secondaire local ne peut pas être supprimé.

Pour les stratégies sur l'utilisation au sein de la limite et la prise des mesures correctives, consultez Taille limite de collection d'éléments.

Tirer profit des index partiellement alloués

Pour tout élément d'une table, DynamoDB écrit une entrée d'index correspondante uniquement si la valeur la clé de tri d'index est présente dans l'élément. Si la clé de tri n'apparaît pas dans chaque élément de table ou si la clé de partition d'index ne figure pas dans l'élément, l'index est considéré comme fragmenté.

Les index fragmentés s'avèrent utiles pour les requêtes sur une petite sous-section d'une table. Par exemple, supposons que vous ayez une table dans laquelle vous stockez toutes vos commandes client avec les attributs de clé suivants :

- Clé de partition: `CustomerId`
- Clé de tri: `OrderId`

Pour suivre les commandes en cours, vous pouvez insérer un attribut nommé `isOpen` dans des articles en commande qui n'ont pas encore été expédiés. Ensuite, lorsque la commande est expédiée, vous pouvez supprimer l'attribut. Si vous créez ensuite un index sur `CustomerId` (clé de partition) et `isOpen` (clé de tri), seules les commandes avec l'attribut `isOpen` défini apparaissent dans celui-ci. Lorsque vous avez des milliers de commandes dont seul un petit nombre est en cours, il est plus rapide et moins cher d'interroger cet index pour rechercher les commandes en cours que d'analyser la totalité de la table.

Au lieu d'utiliser un type d'attribut tel que `isOpen`, vous pourriez utiliser un attribut avec une valeur produisant un ordre de tri utile dans l'index. Par exemple, vous pouvez utiliser un attribut `OrderOpenDate` défini sur la date à laquelle chaque commande a été passée, puis le supprimer une fois que la commande a été exécutée. Ainsi, lorsque vous interrogez l'index fragmenté, les éléments sont renvoyés triés par la date à laquelle la commande a été passée.

Exemples d'index fragmentés dans DynamoDB

Les index secondaires globaux sont fragmentés par défaut. Lorsque vous créez un index secondaire global, vous spécifiez une clé de partition et éventuellement une clé de tri. Seuls les éléments de la table de base qui contiennent ces attributs apparaissent dans l'index.

En définissant un index secondaire global comme étant fragmenté, vous pouvez l'allouer avec un débit d'écriture moins élevé que celui de la table de base, tout en atteignant d'excellentes performances.

Par exemple, une application de jeu peut suivre les scores de chaque utilisateur, mais n'a besoin généralement d'interroger que quelques scores élevés. La conception suivante gère efficacement ce scénario :

Table

Primary Key		Data Attributes...		
Partition Key	*Sort Key*			
Player_ID	Game_ID	Attribute 1	Attribute 2	Attribute 3
Rick	Game_1	Score: 36,750 *(game score)*	Date: 2017-11-14 *(date of game)*	
	Game_2	Score: 69,450 *(game score)*	Date: 2017-12-31 *(date of game)*	
	Game_3	Score: 135,900 *(game score)*	Date: 2018-01-19 *(date of game)*	Award: Champ *(type of award)*
Padma	Game_4	Score: 25,350 *(game score)*	Date: 2018-01-27 *(date of game)*	
	Game_5	Score: 69,450 *(game score)*	Date: 2028-01-19 *(date of game)*	
	Game_6	Score: 147,300 *(game score)*	Date: 2018-02-02 *(date of game)*	Award: Champ *(type of award)*
	Game_7	Score: 169,100 *(game score)*	Date: 2018-03-10 *(date of game)*	Award: Champ *(type of award)*

Ici, Rick a joué trois parties et a atteint le statut `Champ` dans l'une d'entre elles. Padma a joué quatre parties et a atteint le statut `Champ` dans deux d'entre elles. Notez que l'attribut `Award` est présent uniquement dans les éléments où l'utilisateur a remporté une récompense. L'index secondaire global associé ressemble à ce qui suit :

GSI	Primary Key *Partition Key*	Projected Attributes...			
	Award	Player_ID	Game_ID	Score	Date
	Champ	Rick	Game_3	135,900	2018-01-19
		Padma	Game_6	147,300	2018-02-02
		Padma	Game_7	169,100	2018-03-10

L'index secondaire global contient uniquement les scores élevés qui sont fréquemment interrogés, ce qui représente un petit sous-ensemble des éléments de la table de base.

Utilisation d'index secondaire globaux pour les requêtes de regroupement matérialisé dans DynamoDB

La gestion des regroupements et des métriques clés presque en temps réel sur des données qui changent rapidement s'avère de plus en plus utile que les entreprises puissent prendre des décisions rapides. Par exemple, une bibliothèque musicale peut souhaiter mettre en avant ses morceaux les plus téléchargés presque en temps réel.

Examinez la disposition de table de bibliothèque musicale suivante :

Music Library Table

Primary Key		Data-Item Attributes...			
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	
Song-129 *(song ID)*	Details	Title: Wild Love *(song title)*	Artist: Argyboots *(artist or band name)*	Downloads: 15,314,822 *(lifetime total downloads)*	...etc.
	Month-2018-01	GSI Primary Key Month: 2018-01 *(download month)*	GSI Secondary Key MonthTotal: 1,746,992 *(month total downloads)*		
	Dld-9349823681	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823682	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823683	Time: 2018-01-01T00:00:07 *(download timestamp)*			

La table de cet exemple stocke des chansons avec `songID` comme clé de partition. Vous pouvez activer Amazon DynamoDB Streams sur cette table et attacher une fonction Lambda aux flux de sorte que, quand chaque chanson est téléchargée, une entrée soit ajoutée à la table avec

`Partition-Key=SongID` et `Sort-Key=DownloadID`. À mesure que ces mises à jour sont effectuées, elles déclenchent une fonction Lambda dans DynamoDB Streams. La fonction Lambda peut agréger et regrouper les téléchargements par `songID`, et mettre à jour l'élément de niveau supérieur, `Partition-Key=songID`, ainsi que `Sort-Key=Month`. Gardez à l'esprit que si une exécution lambda échoue juste après avoir écrit la nouvelle valeur agrégée, elle peut être relancée et agréger la valeur plusieurs fois, vous laissant avec une valeur approximative.

Pour lire les mises à jour presque en temps réel avec une latence de l'ordre de quelques millisecondes, utilisez l'index secondaire global avec les conditions de requête `Month=2018-01`, `ScanIndexForward=False`, `Limit=1`.

Comme autre optimisation de clé utilisée ici, l'index secondaire global est un index fractionné et est disponible uniquement sur les éléments qui doivent être interrogés pour récupérer les données en temps réel. L'index secondaire global peut traiter des flux de travail supplémentaires qui ont besoin d'informations sur les 10 chansons les plus populaires, ou toute chanson téléchargée au cours de ce mois.

Surcharge des index secondaires globaux dans DynamoDB

Même si Amazon DynamoDB dispose d'un quota par défaut de 20 index secondaires globaux par table, en pratique, vous pouvez indexer beaucoup plus que 20 champs de données. Contrairement à une table dans un système de gestion de base de données relationnelle (SGBDR) où le schéma est uniforme, une table dans DynamoDB peut contenir de nombreux types d'éléments de données différents en même temps. De plus, le même attribut dans des éléments différents peut contenir des types d'informations complètement différents.

Prenez l'exemple suivant d'une disposition de table DynamoDB qui enregistre différents types de données.

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 *(employee ID)*	Employee_Name	Data: Murphy, John *(employee name)*	Start: 2008-11-08 *(start date)*	...etc.
	YYYY-Q1	Data: $5,477 *(order totals in USD)*	Name: Murphy, John *(employee name)*	
	HR_confidential	Data: 2008-11-08 *(hire date)*	Name: Murphy, John *(employee name)*	...etc.
	Warehouse_01	Data: Murphy, John *(employee name)*		
	v0_Job_title	Data: Operator-1 *(job title)*	Start: 2008-11-08 *(start date)*	...etc.
	v1_Job_title	Data: Operator-2 *(job title)*	Start: 2016-11-04 *(start date)*	...etc.
	v2_Job_title	Data: Supervisor-1 *(job title)*	Start: 2017-11-01 *(start date)*	...etc.

L'attribut `Data`, qui est commun à tous les éléments, a un contenu différent selon son élément parent. Si vous créez un index secondaire global pour la table, qui utilise la clé de tri de la table comme clé de partition et l'attribut `Data` comme clé de tri, vous pouvez exécuter tout un éventail de requêtes différentes à l'aide d'un seul index secondaire global. Ces requêtes peuvent comprendre notamment les requêtes suivantes :

- Recherchez un employé par son nom dans l'index secondaire global en utilisant `Employee_Name` comme valeur de clé de partition, et le nom de l'employé (par exemple, `Murphy, John`) comme valeur de clé de tri.
- Utiliser l'index secondaire global pour trouver tous les employés qui travaillent dans un entrepôt particulier en effectuant une recherche sur un ID d'entrepôt (par exemple, `Warehouse_01`).

- Obtenez la liste des embauches récentes en interrogeant l'index secondaire global sur `HR_confidential` comme une valeur de clé de partition et en utilisant une plage de dates en tant que valeur de clé de tri.

Utilisation d'un partitionnement d'écriture d'index secondaire global pour des requêtes de table sélectives dans DynamoDB

Lorsque vous devez interroger des données récentes dans un intervalle de temps spécifique, l'exigence pour DynamoDB de fournir une clé de partition pour la plupart des opérations de lecture peut constituer un défi. Pour résoudre ce scénario, vous pouvez mettre en œuvre un modèle de requête efficace en combinant le partitionnement en écriture et un index secondaire global (GSI).

Cette approche vous permet d'extraire et d'analyser efficacement les données urgentes sans effectuer une analyse complète des tables, qui peut être gourmande en ressources et coûteuses. En concevant de manière stratégique la structure et l'indexation de votre table, vous pouvez créer une solution flexible qui prend en charge l'extraction de données basée sur le temps, tout en maintenant des performances optimales.

Rubriques

- Conception de modèle
- Stratégie de partition
- Interrogation du GSI partitionné
- Considérations relatives à l'exécution parallèle des requêtes
- Exemple de code

Conception de modèle

Lorsque vous utilisez DynamoDB, vous pouvez surmonter les difficultés liées à l'extraction de données basée sur le temps en mettant en œuvre un modèle sophistiqué qui combine le partitionnement en écriture et les index secondaires globaux, afin de permettre des requêtes flexibles et efficaces dans les fenêtres de données récentes.

Structure de la table

- Clé de partition (PK) : « Nom d'utilisateur »

Structure du GSI

- Clé de partition GSI (PK_GSI) : « # » ShardNumber
- Clé de tri GSI (SK_GSI) : horodatage ISO 8601 (par exemple, « 2030-04-01T12:00:00Z »)

[TABLE] - TimeBounded Metadata Actions

PK	SK_GSI	Address	PK_GSI
Trenton69	2023-12-05T06:40:31.312Z	117 Hudson Divide	ShardNumber#0
Scot_Langworth-Prosacco	2024-10-09T14:44:45.819Z	84021 Herzog Skyway	ShardNumber#6
Juliet.Morissette	2025-03-28T07:20:56.538Z	4871 N Broadway	ShardNumber#4
Kay.Von71	2024-11-23T16:19:26.763Z	14554 Odell Throughway	ShardNumber#2
Kiarra43	2023-11-15T16:21:57.078Z	8471 London Road	ShardNumber#1
Marcella91	2024-12-17T09:02:31.623Z	10271 Nick Crescent	ShardNumber#6
Bernard.Lemke36	2025-09-09T00:18:34.482Z	5438 Douglas Groves	ShardNumber#2
Daisha83	2024-01-24T06:38:08.482Z	8786 Armstrong Radial	ShardNumber#3
Marcos_Schmitt58	2024-04-06T17:38:58.551Z	510 S Center Street	ShardNumber#3
Jeremie_VonRueden-Mills	2025-04-06T04:24:23.701Z	157 Koch Drives	ShardNumber#1
Verna28	2025-10-10T18:42:21.059Z	70040 Baylee Trafficway	ShardNumber#1
Trycia.Doyle	2024-01-10T17:00:08.360Z	9511 Tillman Extensions	ShardNumber#2

Stratégie de partition

En supposant que vous décidiez d'utiliser 10 partitions, le nombre de vos partitions peut être compris entre 0 et 9. Lorsque vous enregistrez une activité, vous devez calculer le nombre de partitions (par exemple, en utilisant une fonction de hachage sur l'ID utilisateur, puis en prenant le module du nombre de partitions) et l'ajouter à la clé de partition GSI. Cette méthode répartit les entrées sur différentes partitions, réduisant ainsi le risque de partition chaude.

Interrogation du GSI partitionné

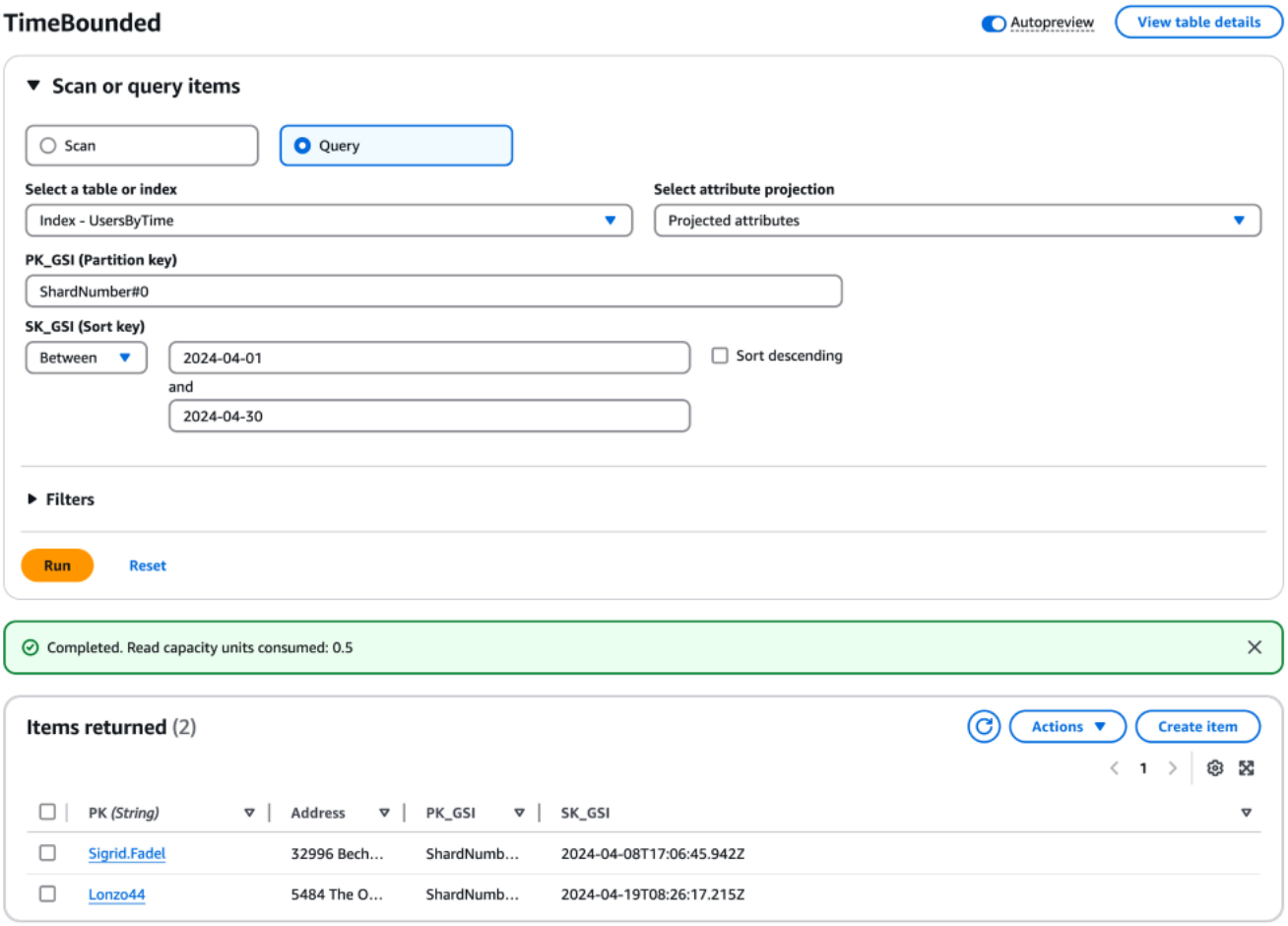
L'interrogation de toutes les partitions à la recherche d'éléments se situant dans un intervalle de temps donné dans une table DynamoDB, où les données sont réparties entre plusieurs clés de partition, nécessite une approche différente de celle utilisée pour interroger une seule partition. Les requêtes DynamoDB étant limitées à une seule clé de partition à la fois, vous ne pouvez pas interroger directement plusieurs partitions en une seule opération de requête. Cependant, vous

pouvez obtenir le résultat souhaité grâce à une logique au niveau de l'application en effectuant plusieurs requêtes, chacune ciblant une partition spécifique, puis en agrégeant les résultats. La procédure ci-dessous explique comment procéder.

Pour interroger et agréger des partitions

1. Identifiez la plage de numéros de partition utilisée dans votre stratégie de mise en partition. Par exemple, si vous avez 10 partitions, le nombre de vos partitions sera compris entre 0 et 9.
2. Pour chaque partition, construisez et exécutez une requête pour récupérer des éléments dans la plage de temps souhaitée. Ces requêtes peuvent être exécutées en parallèle pour améliorer l'efficacité. Utilisez la clé de partition avec le numéro de partition et la clé de tri, ainsi que votre plage de temps, pour ces requêtes. Voici un exemple de requête pour une seule partition :

```
aws dynamodb query \
    --table-name "YourTableName" \
    --index-name "YourIndexName" \
    --key-condition-expression "PK_GSI = :pk_val AND SK_GSI BETWEEN :start_date
 AND :end_date" \
    --expression-attribute-values '{
        ":pk_val": {"S": "ShardNumber#0"},
        ":start_date": {"S": "2024-04-01"},
        ":end_date": {"S": "2024-04-30"}
    }'
```



Vous devez répliquer cette requête pour chaque partition, en ajustant la clé de partition en conséquence (par exemple, "ShardNumber#1 «," ShardNumber #2 «,...," ShardNumber #9 «).

3. Agrégez les résultats de chaque requête une fois que toutes sont terminées. Effectuez cette agrégation dans le code de votre application, en combinant les résultats dans un jeu de données unique qui représente les éléments de toutes les partitions dans la plage de temps spécifiée.

Considérations relatives à l'exécution parallèle des requêtes

Chaque requête consomme la capacité de lecture de votre table ou de votre index. Si vous utilisez le débit provisionné, assurez-vous que votre table est dotée d'une capacité suffisante pour gérer le débordement de requêtes parallèles. Si vous utilisez une capacité à la demande, soyez conscient des implications financières potentielles.

Exemple de code

Pour exécuter des requêtes parallèles sur des partitions dans DynamoDB à l'aide de Python, vous pouvez utiliser la bibliothèque boto3, qui est le kit SDK Amazon Web Services pour Python. Cet exemple suppose que boto3 est installé et configuré avec les informations d'identification appropriées AWS.

Le code Python suivant montre comment effectuer des requêtes parallèles sur plusieurs partitions pour une plage de temps donnée. Il utilise des contrats à terme simultanés pour exécuter des requêtes en parallèle, réduisant ainsi le temps d'exécution global par rapport à une exécution séquentielle.

```
import boto3
from concurrent.futures import ThreadPoolExecutor, as_completed

# Initialize a DynamoDB client
dynamodb = boto3.client('dynamodb')

# Define your table name and the total number of shards
table_name = 'YourTableName'
total_shards = 10  # Example: 10 shards numbered 0 to 9
time_start = "2030-03-15T09:00:00Z"
time_end = "2030-03-15T10:00:00Z"

def query_shard(shard_number):
    """
    Query items in a specific shard for the given time range.
    """
    response = dynamodb.query(
        TableName=table_name,
        IndexName='YourGSIName',  # Replace with your GSI name
        KeyConditionExpression="PK_GSI = :pk_val AND SK_GSI BETWEEN :date_start
 AND :date_end",
        ExpressionAttributeValues={
            ":pk_val": {"S": f"ShardNumber#{shard_number}"},
            ":date_start": {"S": time_start},
            ":date_end": {"S": time_end},
        }
    )
    return response['Items']

# Use ThreadPoolExecutor to query across shards in parallel
```

```
with ThreadPoolExecutor(max_workers=total_shards) as executor:
    # Submit a future for each shard query
    futures = {executor.submit(query_shard, shard_number): shard_number for
 shard_number in range(total_shards)}

    # Collect and aggregate results from all shards
    all_items = []
    for future in as_completed(futures):
        shard_number = futures[future]
        try:
            shard_items = future.result()
            all_items.extend(shard_items)
            print(f"Shard {shard_number} returned {len(shard_items)} items")
        except Exception as exc:
            print(f"Shard {shard_number} generated an exception: {exc}")

# Process the aggregated results (e.g., sorting, filtering) as needed
# For example, simply printing the count of all retrieved items
print(f"Total items retrieved from all shards: {len(all_items)}")
```

Avant d'exécuter ce code, assurez-vous de remplacer `YourTableName` et `YourGSIName` par les noms de table et de GSI de votre configuration DynamoDB. Ajustez également les variables `total_shards`, `time_start` et `time_end` en fonction de vos besoins spécifiques.

Ce script interroge chaque partition à la recherche d'éléments dans la plage de temps spécifiée et agrège les résultats.

Utilisation d'index secondaires globaux pour créer un réplica éventuellement cohérent dans DynamoDB

Vous pouvez utiliser un index secondaire global pour créer un réplica éventuellement cohérent d'une table. La création d'un réplica peut vous permettre d'effectuer les opérations suivantes :

- Définissez une capacité de lecture allouée différente pour différents lecteurs. Supposons par exemple que vous ayez deux applications : une application gère les requêtes hautement prioritaires et a besoin des niveaux de performances de lecture les plus élevés, tandis que l'autre gère les requêtes faiblement prioritaire qui peuvent tolérer une limitation de l'activité de lecture.

  Si ces deux applications lisent à partir de la même table, une lourde charge de lecture de l'application faiblement prioritaire pourrait consommer toute la capacité de lecture disponible pour la table. Cela limiterait l'activité de lecture de l'application hautement prioritaire.

Au lieu de cela, vous pouvez créer un réplica via un index secondaire global pour lequel vous pouvez définir une capacité de lecture distincte de celle de la table elle-même. Vous pouvez alors demander à votre application faiblement prioritaire d'interroger le réplica au lieu de la table.

- Éliminez entièrement les lectures d'une table. Par exemple, vous pouvez avoir une application qui capture un volume élevé d'activité de flux de clics d'un site web, et vous ne voulez pas risquer que des lectures interférent avec cela. Vous pouvez isoler cette table et empêcher les lectures par d'autres applications (voirUtilisation de conditions de politique IAM pour un contrôle d'accès précis), tout en laissant d'autres applications lire un réplica créé à l'aide d'un index secondaire global.

Pour créer un réplica, configurez un index secondaire global ayant le même schéma de clé que la table parent, avec tout ou partie des attributs autres que de clé projetés dans cet index. Dans les applications, vous pouvez diriger une partie ou la totalité de l'activité de lecture vers cet index secondaire global plutôt que vers la table parent. Vous pouvez ensuite ajuster la capacité de lecture approvisionnée de l'index secondaire global pour gérer ces lectures sans modifier la capacité de lecture approvisionnée de la table parent.

Il y a toujours un court délai de propagation entre une écriture dans la table parent et le moment où les données écrites apparaissent dans l'index. En d'autres termes, vos applications doivent tenir compte du fait que le réplica d'index secondaire global n'est qu'éventuellement cohérent avec la table parent.

Vous pouvez créer plusieurs réplicas d'index secondaire global pour prendre en charge différents modèles de lecture. Lorsque vous créez les réplicas, projettez uniquement les attributs dont chaque modèle de lecture a réellement besoin. Une application peut alors consommer moins de capacité de lecture approvisionnée pour obtenir uniquement les données dont elle a besoin au lieu d'avoir à lire l'élément à partir de la table parent. Cette optimisation peut entraîner des économies de coûts considérables au fil du temps.

Bonnes pratiques de stockage d'éléments volumineux et d'attributs dans DynamoDB

Amazon DynamoDB limite à 400 Ko la taille de chaque élément que vous stockez dans une table (voir Quotas dans Amazon DynamoDB). Si votre application a besoin de stocker dans un élément davantage de données que ne le permet la limite de taille dans DynamoDB, vous pouvez essayer de compresser un ou plusieurs attributs volumineux, ou de les scinder en plusieurs éléments (indexés efficacement par des clés de tri). Vous pouvez stocker l'élément en tant qu'objet dans Amazon

Simple Storage Service (Amazon S3), puis stocker l'identifiant d'objet Amazon S3 dans votre élément DynamoDB.

Une bonne pratique consiste à utiliser le paramètre `ReturnConsumedCapacity` lorsque vous rédigez des éléments pour surveiller et signaler les tailles d'éléments approchant la taille d'élément maximale de 400 Ko. Le dépassement de la taille maximale d'élément entraînera l'échec des tentatives d'écriture. DynamoDB renverra une erreur. ValidationException La surveillance et les alertes relatives à la taille des articles vous permettront d'atténuer les problèmes associés avant qu'ils n'aient un impact sur votre demande.

Compression de valeurs d'attribut volumineux

La compression de valeurs d'attribut volumineuses permet de maintenir celles-ci dans les limites de l'élément dans DynamoDB, et de réduire les coûts de stockage. Les algorithmes de compression comme GZIP ou LZO génèrent une sortie binaire que vous pouvez ensuite stocker dans un type d'attribut `Binary` dans l'élément.

Prenons l'exemple d'un tableau qui stocke les messages rédigés par les utilisateurs du forum. Ces messages contiennent souvent de longues chaînes de texte susceptibles d'être compressées. Bien que la compression puisse réduire la taille des éléments, l'inconvénient est que les valeurs d'attributs compressées ne sont pas utiles pour le filtrage.

Pour l'exemple de code qui montre comment compresser de tels messages dans DynamoDB, consultez les ressources suivantes :

- Exemple : gestion d'attributs de type binaire à l'aide de l'API de document de AWS SDK pour Java
- Exemple : gestion d'attributs de type binaire à l'aide de l'API de bas niveau AWS SDK pour .NET

Partitionnement vertical

Une autre solution pour traiter des éléments volumineux consiste à les décomposer en petits blocs de données et à associer tous les éléments pertinents selon la valeur de la clé de partition. Vous pouvez ensuite utiliser une chaîne de clé de tri pour identifier les informations associées stockées à côté. Ce faisant, et en regroupant plusieurs éléments selon la même valeur de clé de partition, vous créez une collection d'éléments.

Pour plus d'informations sur cette approche, consultez :

- Utilisation du partitionnement vertical pour mettre à l'échelle efficacement les données dans Amazon DynamoDB
- Implémentez le partitionnement vertical dans Amazon DynamoDB à l'aide de AWS Glue

Stockage de valeurs d'attribut volumineuses dans Amazon S3

Comme mentionné précédemment, vous pouvez également utiliser Amazon S3 pour stocker des valeurs d'attribut volumineuses qui ne peuvent pas tenir dans un élément DynamoDB. Vous pouvez les stocker en tant qu'objet dans Amazon S3, puis stocker l'identifiant d'objet dans votre élément DynamoDB.

Vous pouvez aussi utiliser la prise en charge des métadonnées d'objet dans Amazon S3 pour fournir un lien vers l'élément parent dans DynamoDB. Stockez la valeur de clé primaire de l'élément en tant que métadonnée Amazon S3 de l'objet dans Amazon S3. Cela facilite souvent la maintenance des objets Amazon S3.

Par exemple, considérez la table `ProductCatalog`. Les éléments de cette table stockent les informations sur le prix de l'article, sa description, les auteurs de livres et les dimensions d'autres produits. Si vous souhaitiez stocker une image de chaque produit trop volumineuse pour tenir dans un élément, vous pourriez stocker les images dans Amazon S3 plutôt que dans DynamoDB.

Lorsque vous implémentez cette politique, gardez à l'esprit les éléments suivants :

- DynamoDB ne prend pas en charge les transactions entre Amazon S3 et DynamoDB. Votre application doit donc gérer les échecs, dont le nettoyage d'objets Amazon S3 orphelins.
- Amazon S3 limite la longueur des identifiants d'objet. Vous devez donc organiser vos données afin d'éviter la génération d'identifiants d'objet de longueur excessive et la violation de toute autre contrainte Amazon S3.

Pour plus d'informations sur l'utilisation d'Amazon S3, consultez le Guide du l'utilisateur Amazon Simple Storage Service.

Bonnes pratiques de gestion des données de séries temporelles dans DynamoDB

Les principes de conception généraux dans Amazon DynamoDB recommandent d'utiliser un nombre minimum de tables. Pour la plupart des applications, une seule table est suffisante. Cependant, pour

les données chronologiques, il sera souvent préférable d'utiliser une table par application et par période.

Modèle de création des données de séries temporelles

Prenons l'exemple d'un scénario classique avec des données chronologiques, dans lequel vous voulez suivre un grand nombre d'événements. Votre modèle d'accès en écriture se résume au fait que tous les événements enregistrés ont la date du jour. Votre modèle d'accès en lecture se résume comme suit : la lecture des événements du jour est la lecture effectuée le plus fréquemment, la lecture des événements d'hier étant effectuée un peu moins fréquemment et celle des événements des jours suivants étant effectuée très rarement. Un moyen de le faire est d'intégrer la date et l'heure actuelles dans la clé primaire.

Le modèle de création suivant permet souvent de gérer de façon efficace ce scénario :

- Créez une table par période, à laquelle vous allouez la capacité en lecture et écriture requise et les index nécessaires.
- Avant la fin de chaque période, générez à l'avance la table pour la période suivante. Lorsque la période en cours se termine, dirigez le trafic d'événements vers la nouvelle table. Vous pouvez affecter à ces tables des noms indiquant les périodes auxquelles elles sont associées.
- Dès qu'une table ne fait plus l'objet d'écritures, réduisez sa capacité en écriture allouée (par exemple 1 WCU) et allouez la capacité en lecture appropriée. Réduisez la capacité en lecture allouée aux tables plus anciennes à mesure qu'elles vieillissent. Vous pouvez choisir d'archiver ou de supprimer les tables dont le contenu est rarement ou jamais nécessaire.

L'idée est d'allouer les ressources nécessaires pour la période en cours, qui présente le plus haut volume de trafic, et de réduire la capacité allouée aux anciennes tables, qui ne sont pas utilisées activement, de manière à économiser de l'argent. En fonction des besoins de votre entreprise, vous devrez peut-être envisager d'écrire le partitionnement de manière à répartir le trafic uniformément selon la clé de partition logique. Pour de plus amples informations, veuillez consulter Utilisation du partitionnement d'écriture pour une répartition équitable des charges de travail dans votre table DynamoDB.

Exemples de tableaux de séries chronologiques

Voici un exemple de série chronologique dans lequel la table actuelle est provisionnée à une read/write capacité supérieure et les anciennes tables sont réduites car elles ne sont pas fréquemment consultées.

Current table — Provisioned at: WCU=750 and RCU=300

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-15	00:00:00.002	17.372 W/Sr	713 nm	...
2018-03-15	00:00:00.004	17.385 W/Sr	712 nm	...
2018-03-15	00:00:00.005	17.478 W/Sr	708 nm	...
2018-03-15	00:00:00.007	19.172 W/Sr	674 nm	...
. . .	. . .	. . .	. . .	...

Previous table — Provisioned at: WCU=1 and RCU=100

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-14	00:00:00.001	16.473 W/Sr	512	...
2018-03-14	00:00:00.003	16.489 W/Sr	519	...
2018-03-14	00:00:00.004	16.814 W/Sr	522	...
2018-03-14	00:00:00.006	16.719 W/Sr	506	...
. . .	. . .	. . .	. . .	...

Older table — Provisioned at: WCU=1 and RCU=1

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-10	00:00:00.001	13.669 W/Sr	456	...
2018-03-10	00:00:00.002	13.522 W/Sr	459	...
2018-03-10	00:00:00.004	13.596 W/Sr	457	...
2018-03-10	00:00:00.005	15.721 W/Sr	425	...
. . .	. . .	. . .	. . .	...

Meilleures pratiques pour gérer les many-to-many relations dans les tables DynamoDB

Les listes d'adjacence constituent un modèle de conception utile pour modéliser les many-to-many relations dans Amazon DynamoDB. Plus généralement, elles fournissent une manière de représenter des données de graphique (nœuds et arcs) dans DynamoDB.

Modèle de conception de liste adjacente

Lorsque différentes entités d'une application ont une many-to-many relation entre elles, cette relation peut être modélisée sous forme de liste d'adjacence. Dans ce modèle, toutes les entités de niveau supérieur (soit les nœuds dans le modèle de graphique) sont représentées à l'aide de la clé de partition. Toute relation avec d'autres entités (arcs dans un graphique) est représentée comme un élément de la partition via la définition de l'ID d'entité cible (nœud cible) en tant que valeur de la clé de tri.

Parmi les avantages de ce modèle, on peut citer la duplication minimale des données et les modèles de requête simplifiés pour la recherche de toutes les entités (nœuds) liées à une entité cible (disposant d'un arc vers un nœud cible).

Un système de facturation dans lequel les factures contiennent plusieurs notes constitue un exemple concret pour lequel ce modèle s'avère utile. Une note peut appartenir à plusieurs factures. Dans cet exemple, la clé de partition est `InvoiceID` ou `BillID`. Les partitions `BillID` ont tous les attributs spécifiques aux notes. Les partitions `InvoiceID` ont un élément qui stocke les attributs spécifiques à la facture, et un élément pour chaque `BillID` qui se regroupe dans la facture.

Le schéma se présente comme suit :

Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	`Invoice-92551`	**Inv_ID:** Invoice-92551 *(invoice ID)*	**Dated:** 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Invoice-92552`	**Inv_ID:** Invoice-92552 *(invoice ID)*	**Dated:** 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Bill-4224663`	**Bill_ID:** Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	More attributes of this bill...
	`Bill-4224687`	**Bill_ID:** Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	More attributes of this bill...

En utilisant le schéma précédent, vous pouvez voir que toutes les notes d'une facture peuvent être interrogées à l'aide de la clé primaire sur la table. Pour rechercher toutes les factures contenant une partie d'une note, créez un index secondaire global sur la clé de tri de la table.

Les projections pour l'index secondaire global se présentent comme suit :

GSI	Primary Key *Partition Key*	Projected Attributes...	
	Bill-4224663	**Bill_ID:** Bill-4224663 ***(table primary key)***	Attributes of this bill...
		Inv_ID: Invoice-92551 ***(table primary key)***	Attributes of this bill *in this invoice*..
	Bill-4224687	**Bill_ID:** Bill-4224687 ***(table primary key)***	Attributes of this bill...
		Inv_ID: Invoice-92551 ***(table primary key)***	Attributes of this bill *in this invoice*..
		Inv_ID: Invoice-92552 ***(table primary key)***	Attributes of this bill *in this invoice*..
	Invoice-92551	**Inv_ID:** Invoice-92551 ***(table primary key)***	Attributes of this invoice...
	Invoice-92552	**Inv_ID:** Invoice-92552 ***(table primary key)***	Attributes of this invoice...

Modèle de graphique matérialisé

De nombreuses applications sont construites autour de la compréhension des classements entre pairs, des relations communes entre entités, de l'état de l'entité voisine et d'autres types de flux de travail de style de graphique. Pour ces types d'applications, prenez en compte le modèle de conception de schéma suivant :

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)	Data	GSI PK	Graph Projections
	1	DATE|2|BIRTH	1980-12-19	Hash(Person.Data)	
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	...
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	

	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	

GSI 2	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
	O-N	DATE|2	NodeID	Data	Graph Projections
			2	1980-12-19	...
		DATE|2|BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON|1	NodeID	Data	
			1	John Doe	
		PERSON|1|FRIEND	NodeID		
			5		
		PERSON|5	NodeID	Data	
			5	Jane Smith	
		PERSON|5|FRIEND	NodeID		
			1		
		PLACE|3	NodeID	Data	
			3	UK|England|London	
		PLACE|3|BIRTH	NodeID		
			5		
		PLACE|4	NodeID	Data	
			4	USA|texas|Austin	
		PLACE|4|BIRTH	NodeID		
			1		
		SKILL|6	NodeID	Data	
			6	Java Developer	
			NodeID	Data	
			1	Java Developer|Senior	
		SKILL|7	NodeID	Data	
			7	Guitar	
			NodeID	Data	
			5	Guitar|Advanced	

Le schéma précédent illustre une structure de données de graphique définie par un ensemble de partitions de données contenant les éléments qui définissent les arcs et les nœuds du graphique. Les éléments d'arc contiennent les attributs `Target` et `Type`. Ces attributs sont utilisés dans le cadre d'un nom de clé composite TypeTarget « » pour identifier l'élément dans une partition de la table principale ou dans un deuxième index secondaire global.

Le premier index secondaire global est construit en fonction de l'attribut `Data`. Cet attribut utilise la surcharge d'index secondaire global comme décrit précédemment pour indexer plusieurs types d'attribut différents, principalement `Dates`, `Names`, `Places` et `Skills`. Ici, un index secondaire global indexe effectivement quatre attributs différents.

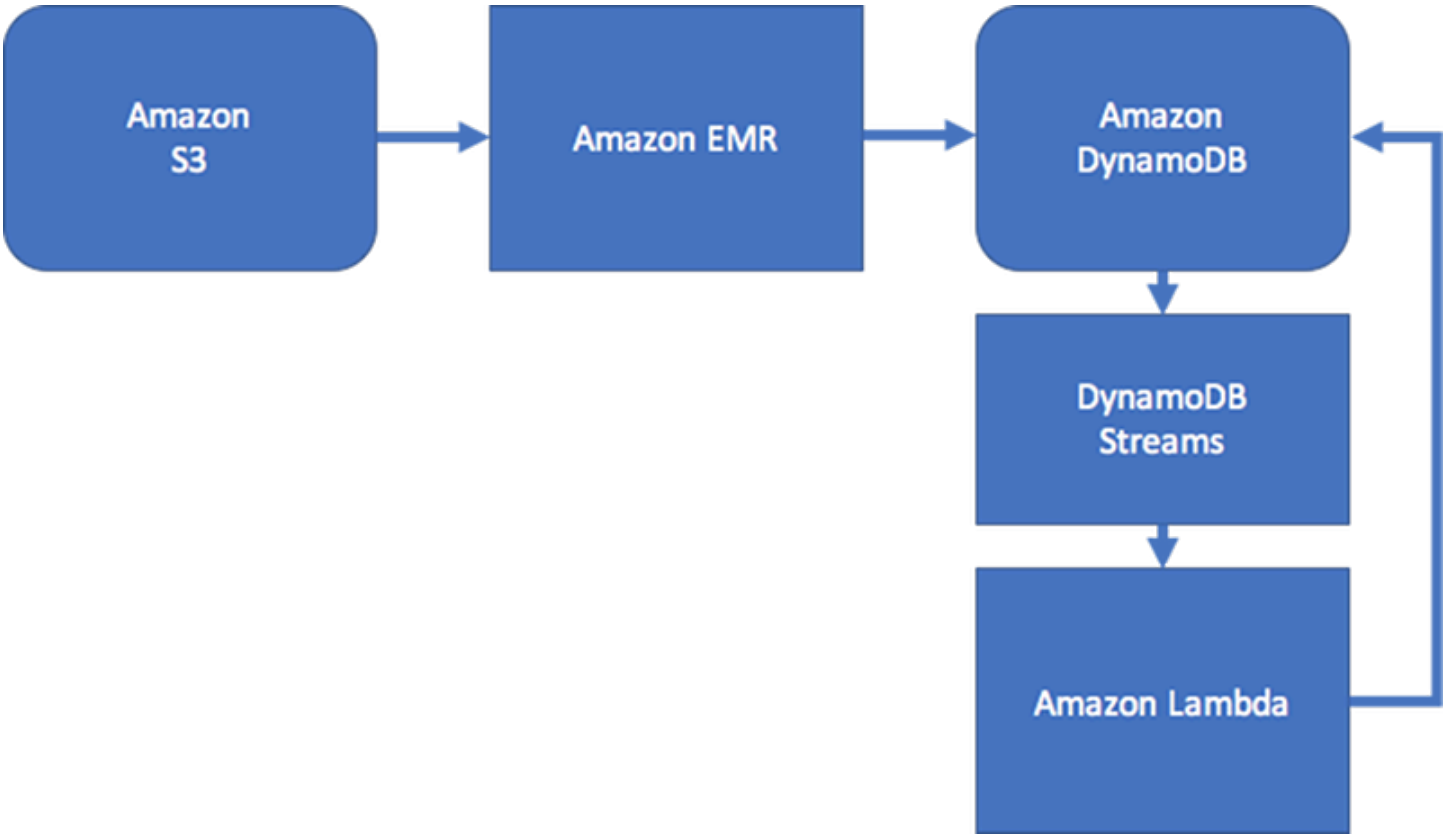
À mesure que vous insérez des éléments dans la table, vous pouvez utilisez une stratégie de partitionnement intelligent pour répartir les ensembles d'éléments avec des regroupements importants (date de naissance, compétence) sur autant de partitions logiques sur les index secondaires globaux que nécessaire pour éviter les problèmes de lecture/écriture à chaud.

Le résultat de cette combinaison de modèles de conception est un magasin de données solide pour des flux de travail de graphiques en temps réel efficaces. Ces flux de travail peuvent fournir un état

de l'entité voisine haute performance et des requêtes de regroupement d'arcs pour des moteurs de recommandations, des classements de nœuds, des regroupements de sous-arborescences, ainsi que d'autres cas d'utilisation de graphiques courants.

Si votre cas d'utilisation n'est pas sensible à la cohérence des données en temps réel, vous pouvez utiliser un processus Amazon EMR planifié pour remplir les arcs avec des regroupements récapitulatifs de graphique pour vos flux de travail. Si votre application n'a pas besoin de connaître immédiatement le moment où un arc est ajouté au graphique, vous pouvez utiliser un processus planifié pour regrouper les résultats.

Pour conserver un certain niveau de cohérence, la conception peut inclure Amazon DynamoDB Streams et AWS Lambda pour le traitement des mises à jour d'arc. Elle peut aussi utiliser une tâche Amazon EMR pour valider les résultats à intervalles réguliers. Le diagramme suivant illustre cette approche. Elle est généralement utilisée dans des applications de réseau social, où le coût d'une requête en temps réel est élevé, et où le besoin de connaître de manière immédiate les mises à jour utilisateur individuelles est faible.



Les applications de gestion de service ITSM et de sécurité ont généralement besoin de répondre en temps réel aux modifications de l'état d'une entité composées de regroupements d'arcs complexe. Ces applications nécessitent un système pouvant prendre en charge des regroupements en temps

réel de plusieurs nœuds présentant des relations de deuxième ou de troisième niveau, ou des traversées d'arcs complexes. Si votre cas d'utilisation nécessite des types de flux de travail de requêtes de graphique en temps réel, nous vous recommandons d'envisager l'utilisation d'Amazon Neptune pour la gestion de ces flux de travail.

> **Note**
>
> Si vous devez interroger des jeux de données hautement connectés ou exécuter des requêtes qui doivent traverser plusieurs nœuds (également nommées requêtes à sauts multiples) avec une latence de quelques millisecondes, vous devriez envisager d'utiliser Amazon Neptune. Amazon Neptune est un moteur de base de données orientée graphe très performant et créé sur mesure, optimisé pour le stockage de milliards de relations et les requêtes de graphe avec une latence de l'ordre de quelques millisecondes.

Bonnes pratiques pour l'interrogation et l'analyse de données dans DynamoDB

Cette section présente certaines bonnes pratiques d'utilisation d'opérations `Query` et `Scan` dans Amazon DynamoDB.

Considérations relatives aux performances pour les analyses

D'une manière générale, les opérations `Scan` sont moins efficaces que d'autres opérations dans DynamoDB. Une opération `Scan` analyse toujours la table entière ou un index secondaire. Elle filtre ensuite les valeurs pour fournir le résultat souhaité, essentiellement en ajoutant l'étape supplémentaire de suppression des données de l'ensemble de résultats.

Si possible, il est recommandé d'éviter d'utiliser une opération `Scan` sur une table ou un index volumineux avec un filtre qui supprime de nombreux résultats. En outre, à mesure qu'une table ou un index augmente, l'opération `Scan` ralentit. L'opération `Scan` examine chaque élément en lien avec les valeurs demandées, et peut utiliser le débit approvisionné pour une table ou un index volumineux en une seule opération. Pour des temps de réponse plus rapides, concevez vos tables et index de façon que vos applications puissent utiliser `Query` au lieu de `Scan`. (Pour les tables, vous pouvez également envisager d'utiliser les `GetItem` touches et `BatchGetItem` APIs.)

Vous pouvez également concevoir votre application pour qu'elle utilise les opérations `Scan` de manière à minimiser l'impact sur votre débit de demandes. Cela peut inclure la modélisation lorsqu'il

peut être plus efficace d'utiliser un index secondaire global plutôt qu'une opération `Scan`. Vous trouverez plus d'informations sur ce processus dans la vidéo suivante.

Modélisation de modèles d'accès à faible vitesse

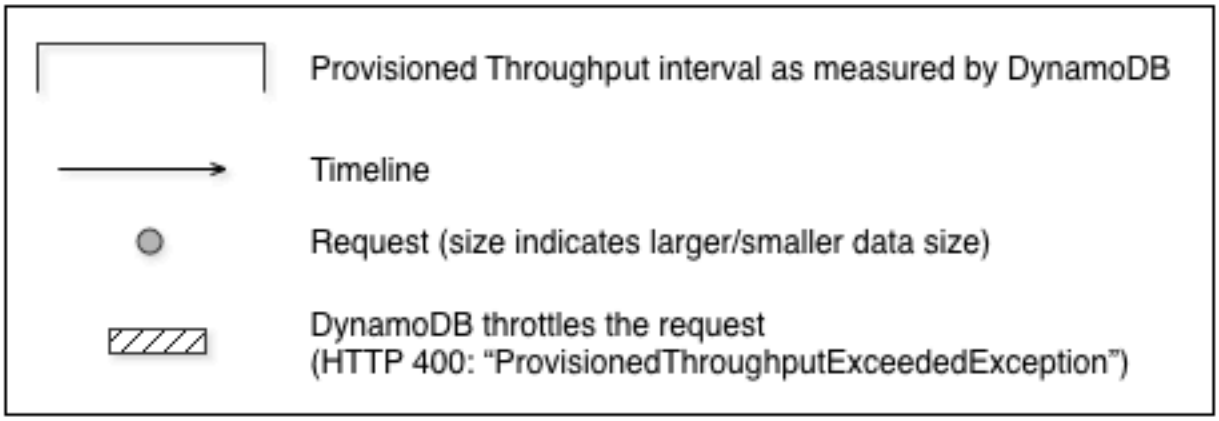
Contournement les pics soudains dans l'activité de lecture

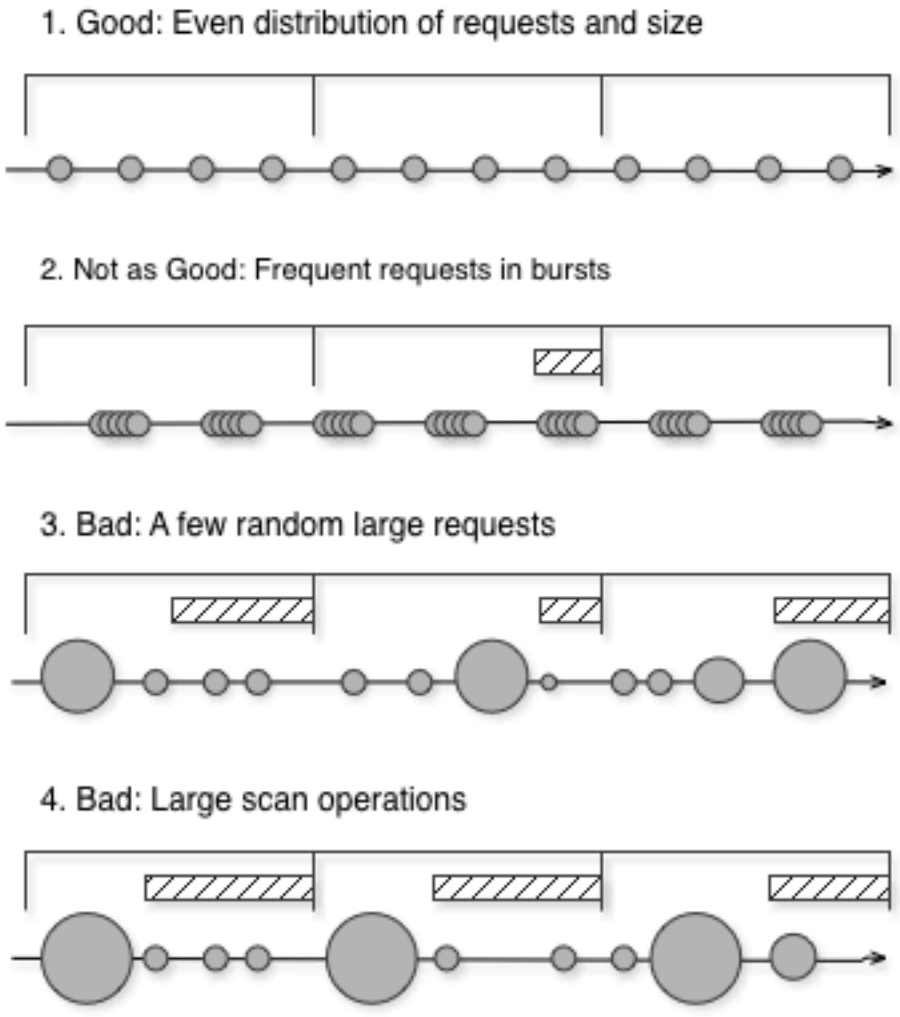
Lorsque vous créez une table, vous définissez ses exigences en termes d'unités de capacité de lecture et d'écriture. Pour les lectures, les unités de capacité sont exprimées en nombre de demandes de lecture de données de 4 Ko fortement cohérentes par seconde. Pour des lectures éventuellement cohérentes, une unité de capacité de lecture correspond à deux demandes de lecture de 4 Ko par seconde. Une opération `Scan` effectue des lectures éventuellement cohérentes par défaut, et peut renvoyer jusqu'à 1 Mo (une page) de données. Par conséquent, une seule demande `Scan` peut consommer (1 Mo de taille de page / 4 Ko de taille d'élément) / 2 (lectures éventuellement cohérentes) = 128 opérations de lecture. Si vous demandez des lectures fortement cohérentes à la place, l'opération `Scan` consomme deux fois plus de débit approvisionné, soit 256 opérations de lecture.

Cela représente un pic soudain d'utilisation par rapport à la capacité de lecture configurée pour la table. Cette utilisation des unités de capacité par une analyse empêche d'autres demandes potentiellement plus importantes pour la même table d'utiliser les unités de capacité disponibles. Par conséquent, il est probable que vous obteniez une exception `ProvisionedThroughputExceeded` pour ces demandes.

Le problème n'est pas seulement l'augmentation soudaine des unités de capacité que l'opération `Scan` utilise. Il est également probable que l'analyse consomme toutes les unités de capacité de la même partition, car elle demande la lecture d'éléments qui se jouxtent sur la partition. Cela signifie que la demande touche la même partition, entraînant la consommation de toutes ses unités de capacité et la limitation des autres demandes adressées à cette partition. Si la demande de lecture des données est répartie sur plusieurs partitions, l'opération n'a pas pour effet de limiter une partition spécifique.

Le diagramme suivant illustre l'impact d'un pic soudain d'utilisation d'unités de capacité par des opérations `Query` et `Scan`, et son incidence sur vos autres demandes par rapport à la même table.





Comme illustré ici, le pic d'utilisation peut avoir un impact sur le débit approvisionné de la table de plusieurs façons :

1. Bon : répartition uniforme des demandes et de la taille

2. Pas très bon : demandes fréquentes en rafales
3. Mauvais : quelques demandes de grande taille aléatoires
4. Mauvais : opérations d'analyse de grande taille

Au lieu d'utiliser une opération `Scan` de grande taille, vous pouvez utiliser les techniques suivantes pour réduire l'impact d'une analyse sur le débit approvisionné d'une table.

- Réduire la taille de page

  Étant donné qu'une opération d'analyse lit une page entière (par défaut, 1 Mo), vous pouvez réduire l'impact de l'opération de numérisation en définissant une taille de page plus petite. L'opération `Scan` fournit un paramètre, Limit, que vous pouvez utiliser pour définir la taille de page pour votre demande. Chaque demande `Query` ou `Scan` portant sur une taille de page plus petite utilise moins d'opérations de lecture et ménage une « pause » entre chaque demande. Par exemple, supposons que chaque élément a une taille de 4 Ko et que vous définissez la taille de page sur 40 éléments. Une demande `Query` ne consomme alors que 20 opérations de lecture éventuellement cohérente ou 40 opérations de lecture fortement cohérente. Un plus grand nombre d'opérations `Query` ou `Scan` de plus petite taille permettrait à vos autres demandes critiques d'aboutir sans limitation.

- Isoler les opérations d'analyse

  DynamoDB est conçu pour faciliter le mise à l'échelle. Par conséquent, une application peut créer des tables à des fins distinctes, voire dupliquer du contenu sur plusieurs tables. Vous souhaitez effectuer des analyses sur une table qui ne prend pas de trafic « stratégique ». Certaines applications gèrent cette charge en opérant une rotation horaire du trafic entre deux tables, l'une pour le trafic critique, et l'autre pour la comptabilisation. D'autres applications peuvent faire cela en effectuant chaque écriture sur deux tables : une table « stratégique » et une table « alternative ».

Configurez votre application pour qu'elle relance toute demande recevant un code de réponse indiquant que vous avez dépassé votre débit approvisionné. Ou augmentez le débit approvisionné pour votre table à l'aide de l'opération `UpdateTable`. Si votre charge de travail comporte des pics temporaires qui ont pour effet que votre débit dépasse parfois le niveau approvisionné, relancez la demande avec un backoff exponentiel. Pour plus d'informations sur l'implémentation d'un backoff exponentiel, consultez Nouvelles tentatives après erreur et backoff exponentiel.

Tirer parti des analyses parallèles

De nombreuses applications peuvent bénéficier de l'utilisation d'opérations `Scan` parallèles au lieu d'analyses séquentielles. Par exemple, une application qui traite une table de données historiques de grande taille peut effectuer une analyse parallèle beaucoup plus rapidement qu'une application séquentielle. Plusieurs unités d'exécution de travail dans un processus de « balayage » en arrière-plan pourraient analyser une table à un faible niveau de priorité sans que cela affecte le trafic de production. Dans chacun de ces exemples, une opération `Scan` parallèle est utilisée de façon à ne pas priver d'autres applications de ressources de débit approvisionné.

Bien que des analyses parallèles puissent être bénéfiques, elles peuvent imposer une forte demande sur le débit approvisionné. Avec une analyse parallèle, votre application dispose de plusieurs unités d'exécution de travail qui exécutent tous des opérations `Scan` simultanément. Cela peut rapidement consommer toute la capacité de lecture approvisionnée de votre table. Dans ce cas, d'autres applications qui doivent accéder à la table risquent d'être limitées.

Une analyse parallèle peut être le bon choix si les conditions suivantes sont réunies :

- La taille de la table est de 20 Go ou plus.
- Le débit de lecture approvisionné de la table n'est pas entièrement utilisé.
- Les opérations `Scan` séquentielles sont trop lentes.

Choisir TotalSegments

Le paramétrage optimal pour `TotalSegments` dépend de vos données spécifiques, des paramètres de débit approvisionné de la table et de vos exigences en matière de performances. Il se peut que vous deviez expérimenter différents paramétrages pour trouver le bon. Nous vous recommandons de commencer par un ratio simple, tel qu'un segment par 2 Go de données. Par exemple, pour une table de 30 Go, vous pouvez définir `TotalSegments` sur 15 (30 Go / 2 Go). Votre application utiliserait alors 15 unités d'exécution de travail, chacun analysant un segment différent.

Vous pouvez également choisir pour `TotalSegments` une valeur basée sur les ressources du client. Vous pouvez définir `TotalSegments` sur n'importe quelle valeur de 1 à 1 000 000, et DynamoDB vous permet d'analyser ce nombre de segments. Par exemple, si votre client limite le nombre d'unités d'exécution pouvant s'exécuter simultanément, vous pouvez augmenter `TotalSegments` progressivement jusqu'à obtenir des performances de `Scan` optimales avec votre application.

Surveillez vos analyses parallèles pour optimiser votre utilisation du débit approvisionné, tout en vous veillant à ce que vos autres applications ne sont pas privées de ressources. Augmentez la valeur de `TotalSegments` si vous ne consommez pas tout votre débit approvisionné mais rencontrez toujours une limitation de vos demandes `Scan`. Réduisez la valeur de `TotalSegments` si les demandes `Scan` consomment plus de débit approvisionné que souhaité.

Bonnes pratiques relatives à la conception d'une table DynamoDB

Les principes de conception généraux dans Amazon DynamoDB recommandent d'utiliser un nombre minimum de tables. Dans la plupart des cas, nous vous recommandons d'utiliser une seule table. Toutefois, si une seule table ou un petit nombre de tables n'est pas viable, ces directives peuvent être utiles.

- La limite par compte ne peut pas être supérieure à 10 000 tables. Si votre application nécessite davantage de tables, prévoyez de les distribuer sur plusieurs comptes. Pour plus d'informations, consultez Quotas de service, de compte et de table dans Amazon DynamoDB.
- Tenez compte des limites du plan de contrôle pour les opérations simultanées du plan de contrôle susceptibles d'avoir un impact sur la gestion de vos tables.
- Travaillez avec des architectes de AWS solutions pour valider vos modèles de conception pour les conceptions multi-locataires.

Utilisation de tables globales DynamoDB

Les tables globales s'appuient sur l'étendue internationale d'Amazon DynamoDB pour vous fournir une base de données entièrement gérée, à régions multiples et à activités multiples, qui peut fournir des performances de lecture et d'écriture rapides et locales, pour des applications dimensionnées massivement et internationales. Les tables globales répliquent automatiquement vos tables DynamoDB parmi celles de votre choix.Régions AWS Aucune modification d'application n'est requise car les tables globales utilisent DynamoDB APIs existant. L'utilisation de tables globales n'entraîne ni frais initiaux ni engagement. Vous ne payez que les ressources que vous utilisez.

Ce guide explique comment utiliser efficacement les tables globales DynamoDB. Il fournit des informations clés sur les tables globales, explique les principaux cas d'utilisation de la fonctionnalité, décrit les deux modes de cohérence, présente une taxonomie de trois modèles d'écriture différents à prendre en compte, passe en revue les quatre principales options de routage des demandes que vous pourriez implémenter, explique comment évacuer une région active ou une région hors ligne,

comment envisager la planification de la capacité de débit et fournit une liste des éléments à prendre en compte lors du déploiement de tables globales.

Ce guide s'inscrit dans le contexte plus large des déploiements AWS multirégionaux, tel que décrit dans le livre blanc sur les principes fondamentaux du AWS multirégional et dans les modèles de conception relatifs à la résilience des données avec vidéo.AWS

Rubriques

- Faits importants sur la conception d’une table globale DynamoDB
- Faits importants à propos de la MREC
- Faits importants à propos de la MRSC
- Cas d’utilisation de table globale MREC DynamoDB
- Modes d’écriture avec des tables globales DynamoDB
- Stratégies de routage dans DynamoDB
- Processus d’évacuation
- Planification de la capacité de débit pour les tables globales DynamoDB
- Liste de vérification pour la préparation des tables globales DynamoDB
- Conclusion et ressources

Faits importants sur la conception d’une table globale DynamoDB

- Il existe deux versions de tables globales : la version actuelle, Tables globales version 2019.11.21 (actuelle), parfois appelée « V2 », et Tables globales de la version 2017.11.29 (héritée) (parfois appelée « V1 »). Ce guide se concentre exclusivement sur la version actuelle.
- DynamoDB (sans tables globales) est un service régional, ce qui signifie qu’il est hautement disponible et intrinsèquement résistant aux défaillances de l’infrastructure, y compris à la défaillance de l’ensemble d’une zone de disponibilité. Une table DynamoDB à région unique est conçue pour offrir une disponibilité de 99,99 %. Pour plus d’informations, consultez DynamoDB service-level agreement (SLA).
- Une table globale DynamoDB réplique ses données entre deux régions ou plus. Une table DynamoDB multi-région est conçue pour offrir une disponibilité de 99,999 %. Une planification adéquate permet aux tables globales d’aider à créer une architecture résiliente face aux défaillances régionales.

- DynamoDB ne possède pas de point de terminaison global. Toutes les demandes sont adressées à un point de terminaison régional, qui accède à l'instance de table globale locale à cette région.
- Les appels à DynamoDB ne doivent pas passer d'une région à une autre. Selon la bonne pratique, une application hébergée dans une région doit accéder directement et uniquement au point de terminaison DynamoDB local de cette région. Si des problèmes sont détectés dans une région (dans la couche DynamoDB ou la pile environnante), le trafic de l'utilisateur final doit être acheminé vers le point de terminaison d'une autre application qui est hébergée dans une autre région. Les tables globales garantissent que l'application hébergée dans chaque région a accès aux mêmes données.

Modes de cohérence

Lorsque vous créez une table globale, vous configurez son mode de cohérence. Les tables globales prennent en charge deux modes de cohérence : la cohérence à terme multi-région (MREC) et la cohérence forte multi-région (MRSC) qui a été introduite en juin 2025.

Si vous ne spécifiez pas de mode de cohérence lorsque vous créez une table globale, celle-ci est définie par défaut sur MREC. Une table globale ne peut pas contenir de réplicas configurés avec différents modes de cohérence. Vous ne pouvez pas modifier le mode de cohérence d'une table globale après sa création.

Faits importants à propos de la MREC

- Les tables globales qui utilisent la MRSC utilisent un modèle de réplication actif-actif. Du point de vue de DynamoDB, la table de chaque région dispose du même statut pour accepter les demandes de lecture et d'écriture. Après avoir reçu une demande d'écriture, la table de réplica locale réplique l'opération d'écriture vers les autres régions participantes distantes en arrière-plan.
- Les éléments sont répliqués individuellement. Les éléments mis à jour au cours d'une même transaction ne peuvent pas être répliqués ensemble.
- Chaque partition de table dans la région source réplique ses opérations d'écriture en parallèle avec toutes les autres partitions. La séquence des opérations d'écriture dans une région distante peut ne pas correspondre à la séquence des opérations d'écriture effectuées dans la région source. Pour plus d'informations sur les partitions de table, consultez le billet de blog Mise à l'échelle de DynamoDB : comment les partitions, les touches de raccourci et les divisions de chaleur ont un impact sur les performances.
- Un élément nouvellement écrit est généralement propagé à toutes les tables de réplica en une seconde. Les régions voisines ont tendance à se propager plus rapidement.

- Amazon CloudWatch fournit une `ReplicationLatency` métrique pour chaque paire de régions. Elle est calculée selon les éléments qui arrivent, la comparaison de leur heure d'arrivée avec leur temps d'écriture initial, et le calcul d'une moyenne. Les horaires sont enregistrés CloudWatch dans la région source. L'affichage des délais moyen et maximum peut aider à déterminer le délai de réplication moyen et celui dans le pire des cas. Il n'existe aucun SLA sur cette latence.
- Si un élément est mis à jour à peu près au même moment (dans cette fenêtre de `ReplicationLatency`) dans deux régions différentes et que la deuxième opération d'écriture a lieu avant que la première ne soit répliquée, il existe un risque de conflit d'écriture. Les tables globales qui utilisent la MREC résolvent ces conflits grâce à un mécanisme de victoire du dernier auteur basé sur l'horodatage des opérations d'écriture. La première opération « perd » face à la seconde. Ces conflits ne sont pas enregistrés dans CloudWatch ou AWS CloudTrail.
- Chaque élément possède un horodatage de la dernière écriture conservé en tant que propriété système privée. L'approche de victoire du dernier auteur est mise en œuvre en utilisant une opération d'écriture conditionnelle qui exige que l'horodatage de l'élément entrant soit ultérieur à l'horodatage de l'élément existant.
- Une table globale réplique tous les éléments dans toutes les régions participantes. Si vous souhaitez disposer de différentes étendues de réplication, vous pouvez créer plusieurs tables globales et attribuer à chacune des tables des régions participantes différentes.
- Les régions locales acceptent les opérations d'écriture même si la région de réplica est hors ligne ou si `ReplicationLatency` augmente. La table locale continue de tenter de répliquer les éléments vers la table distante jusqu'à ce que chaque élément réussisse.
- Dans le cas peu probable où une région est complètement hors ligne, toutes les réplications sortantes et entrantes en attente sont retentées lors de sa reconnexion ultérieurement. Aucune action particulière n'est requise pour rétablir la synchronisation des tables. Le mécanisme de victoire du dernier auteur garantit que les données finiront par devenir cohérentes.
- Vous pouvez ajouter une nouvelle région à une table MREC DynamoDB à tout moment. DynamoDB se charge de la synchronisation initiale et de la réplication en cours. Vous pouvez également supprimer une région (même celle d'origine), ce qui supprimera la table locale de cette région.

Faits importants à propos de la MRSC

- Les tables globales qui utilisent la MRSC utilisent un modèle de réplication actif-actif. Du point de vue de DynamoDB, la table de chaque région dispose du même statut pour accepter les demandes de lecture et d'écriture. Les modifications d'éléments dans un réplica de table globale MRSC sont

répliquées de manière synchrone dans au moins une autre région avant que l'opération d'écriture ne renvoie une réponse réussie.

- Les opérations de lecture fortement cohérente sur tout réplica MRSC renvoient toujours la dernière version d'un élément. Les opérations d'écriture conditionnelle évaluent toujours l'expression de condition par rapport à la dernière version d'un élément. Les mises à jour s'appliquent toujours à la dernière version d'un élément.
- Les opérations de lecture cohérente à terme sur un réplica MRSC peuvent ne pas inclure les modifications récemment survenues dans une autre région, et peuvent même ne pas inclure les modifications survenues très récemment dans la même région.
- Une opération d'écriture échoue avec une exception `ReplicatedWriteConflictException` lorsqu'elle tente de modifier un élément déjà en cours de modification dans une autre région. Les opérations d'écriture qui échouent avec l'exception `ReplicatedWriteConflictException` peuvent être relancées. Elles réussissent si l'élément n'est plus modifié dans une autre région.
- Avec la MRSC, les latences sont plus élevées pour les opérations d'écriture et pour les opérations de lecture fortement cohérente. Ces opérations nécessitent une communication interrégionale. Cette communication peut ajouter une latence qui augmente en fonction de la latence aller-retour entre la région à laquelle vous accédez et celle la plus proche figurant dans la table globale. Pour plus d'informations, consultez la présentation AWS re:Invent 2024, Multi-Region strong consistency with DynamoDB global tables. Les opérations de lecture cohérente à terme ne subissent aucune latence supplémentaire. Il existe un outil de test open source qui vous permet de calculer de manière expérimentale ces latences avec vos régions.
- Les éléments sont répliqués individuellement. Les tables globales utilisant MRSC ne prennent pas en charge la transaction. APIs
- Une table globale MRSC doit être déployée dans exactement trois régions. Vous pouvez configurer une table globale MRSC avec trois réplicas, ou avec deux réplicas et un témoin. Un témoin est un composant d'une table globale MRSC qui contient des données récentes écrites dans des réplicas de table globale. Un témoin fournit une alternative optionnelle à un réplica complet, tout en prenant en charge l'architecture de disponibilité de la MRSC. Vous ne pouvez pas effectuer d'opérations de lecture ou d'écriture sur un témoin. Un témoin n'est pas soumis à des frais de stockage ni d'écriture. Un témoin se trouve dans une région différente de celle des deux réplicas.
- Pour créer une table globale MRSC, ajoutez un réplica et un témoin, ou ajoutez deux réplicas à une table DynamoDB existante qui ne contient aucune donnée. Vous ne pouvez pas ajouter de réplicas supplémentaires à une table globale MRSC existante. Vous ne pouvez pas supprimer un seul réplica ou un seul témoin d'une table globale MRSC. Vous pouvez supprimer deux réplicas,

ou supprimer un réplica et un témoin, d’une table globale MRSC. Le second scénario convertit le réplica restant en une table DynamoDB à région unique.

- Vous pouvez déterminer si un témoin est configuré dans une table globale MRSC, et dans quelle région elle est configurée, à partir de la sortie de l' DescribeTable API. Le témoin est détenu et géré par DynamoDB et n'apparaît pas dans Compte AWS votre région dans laquelle il est configuré.
- Les tables globales MRSC sont disponibles dans les ensembles de régions suivants :
  - Groupe de régions États-Unis : USA Est (Virginie du Nord), USA Est (Ohio), USA Ouest (Oregon)
  - Groupe de régions UE : Europe (Francfort), Europe (Irlande), Europe (Londres), Europe (Paris)
  - Groupe de régions AP : Asie-Pacifique (Tokyo), Asie-Pacifique (Séoul) et Asie-Pacifique (Osaka)
- Les tables globales MRSC ne peuvent pas couvrir des groupes de régions. Par exemple, une table globale MRSC ne peut pas contenir de réplicas provenant à la fois de groupes de régions des États-Unis et de l’UE.
- La durée de vie (TTL) n’est pas prise en charge pour les tables globales MRSC.
- Les index secondaires locaux (LSIs) ne sont pas pris en charge pour les tables globales MRSC.
- CloudWatch Les informations de Contributor Insights ne sont communiquées que pour la région dans laquelle une opération a eu lieu.
- La région locale accepte toutes les opérations de lecture et d’écriture à condition qu’une deuxième région héberge un réplica ou un témoin pour établir le quorum. Si une deuxième région n’est pas disponible, la région locale peut uniquement proposer des lectures cohérentes à terme.
- Dans le cas peu probable où une région serait complètement hors ligne, elle rattrapera automatiquement son retard lorsqu’elle sera de nouveau en ligne. Jusqu'à ce que le problème soit rattrapé, les opérations d'écriture et les opérations de lecture très cohérentes effectuées uniquement dans la région de rattrapage renverront des erreurs tandis que les demandes adressées aux autres régions continueront de fonctionner normalement. À terme, des opérations de lecture cohérentes vers la région de rattrapage renverront les données qui ont été propagées jusqu'à présent dans la région, avec le comportement de cohérence local habituel entre le nœud principal et les répliques locales. Aucune action particulière n’est requise pour rétablir la synchronisation des tables.

Cas d’utilisation de table globale MREC DynamoDB

Les tables globales MREC offrent les avantages suivants :

- Opérations de lecture à faible latence. Placez une copie des données plus près de l'utilisateur final afin de réduire la latence du réseau lors des opérations de lecture. Les données sont conservées aussi longtemps que la valeur `ReplicationLatency`.
- Opérations d'écriture à faible latence. Vous pouvez écrire dans une région voisine pour réduire la latence du réseau et le temps nécessaire à l'écriture. Le trafic d'écriture doit être acheminé avec soin pour éviter tout conflit. Les techniques de routage sont détaillées dans Stratégies de routage dans DynamoDB.
- Migration fluide entre les régions. Vous pouvez ajouter une nouvelle région, puis supprimer l'ancienne région pour migrer un déploiement d'une région vers une autre, le tout sans durée d'indisponibilité au niveau de la couche de données.

Les tables globales MREC et MRSC offrent toutes deux cet avantage :

- Résilience et reprise après sinistre accrues. Si une région présente une dégradation des performances ou une panne complète, vous pouvez l'évacuer. Évacuer signifie rejeter une partie ou la totalité des demandes adressées à cette région. L'utilisation de tables globales augmente également le SLA de DynamoDB pour le pourcentage de durée de fonctionnement mensuel de 99,99 % à 99,999 %. L'utilisation de la MREC prend en charge un objectif de point de reprise (RPO) et un objectif de délai de reprise (RTO) mesurés en secondes. L'utilisation de la MRSC permet d'obtenir un RPO nul.

  Par exemple, Fidelity Investments a présenté lors de re:Invent 2022 la façon dont elle utilise les tables globales DynamoDB pour son système de gestion des commandes. Son objectif était de parvenir à un traitement fiable et à faible latence avec une échelle qu'un traitement sur site ne pourrait pas atteindre, tout en préservant la résilience face aux défaillances régionales et dans les zones de disponibilité.

Si votre objectif est la résilience et la reprise après sinistre, les tables MRSC présentent des latences supérieures d'écriture et de lecture fortement cohérentes, mais elles atteignent un RPO nul. Les tables globales MREC prennent en charge un RPO égal au délai de réplication entre les réplicas, généralement de quelques secondes selon les régions du réplica.

Modes d'écriture avec des tables globales DynamoDB

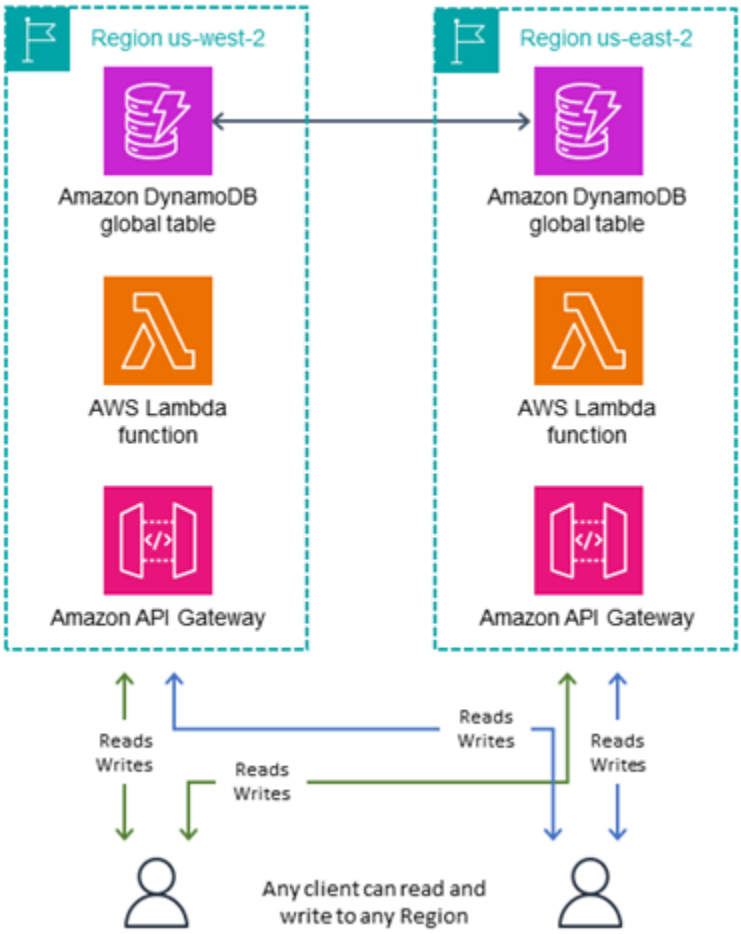
Les tables globales sont toujours actives-actives au niveau de la table. Toutefois, en particulier pour les tables MREC, vous pouvez les traiter comme actives-passives en contrôlant la façon dont vous acheminez les demandes d'écriture. Par exemple, vous pouvez décider d'acheminer les demandes

d'écriture vers une seule région afin d'éviter les éventuels conflits d'écriture qui peuvent se produire dans les tables MREC.

Il existe trois principaux modèles d'écriture gérés, comme l'expliquent les trois sections suivantes. Vous devez déterminer quel modèle d'écriture correspond à votre cas d'utilisation. Ce choix affecte la manière dont vous acheminez les demandes, évacuez une région et gérez la reprise après sinistre. Les instructions dans les sections suivantes dépendent du mode d'écriture de votre application.

Mode Écrire dans n'importe quelle région (non primaire)

Le mode Écrire dans n'importe quelle région, illustré dans le diagramme suivant, est entièrement actif-actif et n'impose aucune restriction quant à l'endroit où une écriture peut avoir lieu. Toute région peut accepter une écriture à tout moment. Il s'agit du mode le plus simple, mais il ne peut être utilisé qu'avec certains types d'applications. Ce mode convient à toutes les tables MRSC. Il convient également pour les tables MREC lorsque tous les rédacteurs sont idempotents et peut donc être répété en toute sécurité afin que les opérations d'écriture simultanées ou répétées entre les régions ne soient pas en conflit. Par exemple, lorsqu'un utilisateur met à jour ses coordonnées. Ce mode fonctionne également bien dans un cas particulier d'idempotent, à savoir un jeu de données à ajout uniquement où toutes les écritures sont des insertions uniques sous une clé primaire déterministe. Enfin, ce mode convient pour la MREC lorsque le risque d'écritures conflictuelles est acceptable.



Le mode Écrire dans n'importe quelle région est l'architecture la plus simple à implémenter. Le routage est plus facile car n'importe quelle région peut être la cible d'écriture à tout moment. Le basculement est plus facile, car avec les tables MRSC, les éléments sont toujours synchronisés et toutes les écritures récentes peuvent être rejouées autant de fois que vous le souhaitez dans n'importe quelle région secondaire. Dans la mesure du possible, votre conception doit suivre ce mode d'écriture.

Par exemple, plusieurs services de streaming vidéo utilisent des tables globales pour suivre les favoris, les avis, les indicateurs du statut des visionnages, etc. Ces déploiements utilisent des tables MREC, car ils ont besoin de réplicas dispersés dans le monde entier, chaque réplica fournissant des opérations de lecture et d'écriture à faible latence. Ces déploiements peuvent utiliser l'écriture dans n'importe quelle région, à condition de s'assurer que chaque opération d'écriture est idempotente. C'est le cas si chaque mise à jour (par exemple, la définition d'un nouveau code horaire, l'attribution d'un nouvel avis ou la définition d'un nouveau statut de surveillance) attribue directement le nouvel état à l'utilisateur, et si la prochaine valeur correcte pour un article ne dépend pas de sa valeur actuelle. Si les demandes d'écriture de l'utilisateur sont acheminées vers différentes régions, la

dernière opération d'écriture persiste et l'état global est réglé en fonction de la dernière attribution. Les opérations de lecture dans ce mode finissent par devenir cohérentes, après avoir été retardées par la dernière valeur de `ReplicationLatency`.

Autre exemple : une entreprise de services financiers utilise des tables globales dans le cadre d'un système visant à tenir à jour le décompte des achats par carte de débit pour chaque client, afin de calculer les remises en argent de ce client. Elle veut conserver un article `RunningBalance` par client. Ce mode d'écriture n'est pas naturellement idempotent, car au fur et à mesure que les transactions entrent, elles modifient l'équilibre en utilisant une expression `ADD` dans laquelle la nouvelle valeur correcte dépend de la valeur actuelle. En utilisant les tables MRSC, elles peuvent toujours écrire dans n'importe quelle région, car chaque appel `ADD` fonctionne toujours en fonction de la toute dernière valeur de l'article.

Un troisième exemple concerne une entreprise qui fournit des services de placement d'annonces en ligne. Cette entreprise a décidé qu'un faible risque de perte de données serait acceptable pour simplifier la conception du mode Écrire dans n'importe quelle région. Lorsqu'elle diffuse des publicités, elle ne dispose que de quelques millisecondes pour extraire suffisamment de métadonnées afin de déterminer la publicité à diffuser, puis enregistrer l'impression publicitaire afin que celle-ci ne soit pas répétée trop rapidement. Elle utilise des tables globales pour obtenir à la fois des opérations de lecture à faible latence pour les utilisateurs finaux du monde entier et des opérations d'écriture à faible latence. Elle enregistre toutes les impressions publicitaires d'un utilisateur au sein d'un seul élément, qui est représenté sous la forme d'une liste croissante. Elle utilise un seul élément au lieu de l'ajouter à une collection d'éléments, car elle peut ainsi supprimer les anciennes impressions publicitaires à chaque opération d'écriture sans avoir à payer une opération de suppression. Cette opération d'écriture n'est pas idempotente : si le même utilisateur final voit des publicités diffusées dans plusieurs régions à peu près au même moment, il est possible qu'une opération d'écriture pour une impression publicitaire remplace l'autre. Le risque est qu'un utilisateur voie une publicité se répéter de temps en temps. L'entreprise a décidé que cette situation était acceptable.

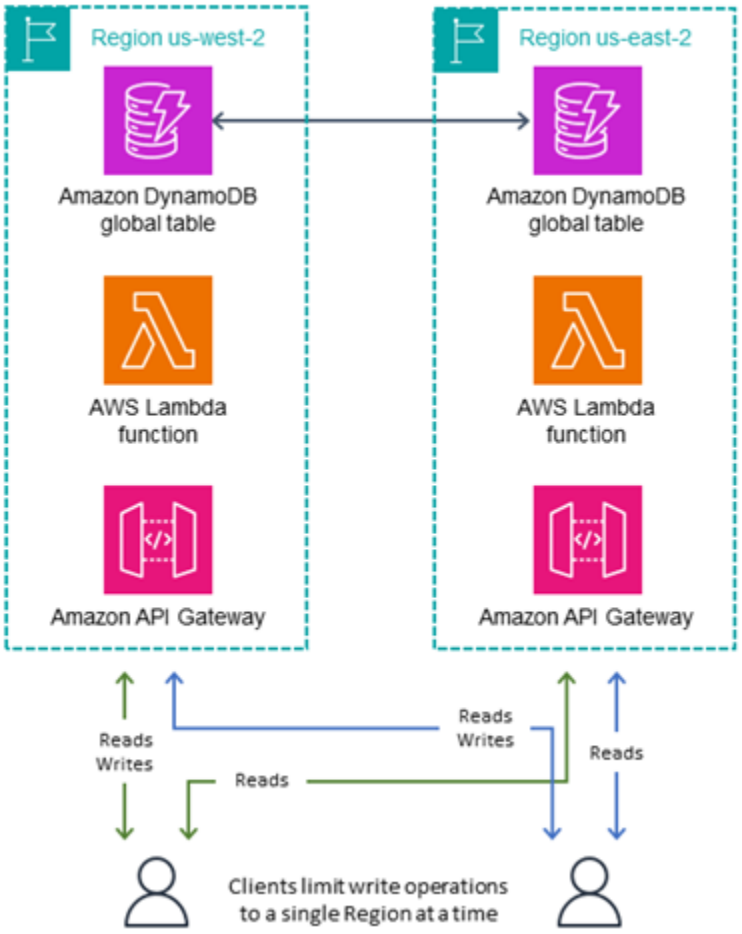
Écrire dans une région (primaire unique)

Le mode Écrire dans une région, illustré dans le diagramme suivant, est actif-passif et achemine toutes les écritures de table vers une seule région active. Notez que DynamoDB n'a pas la notion de région active unique ; le routage des applications en dehors de DynamoDB gère cela. Le mode Écrire dans une région est efficace pour les tables MREC qui doivent éviter les conflits d'écriture, en garantissant que les opérations d'écriture ne circulent que vers une région à la fois. Ce mode d'écriture est utile lorsque vous souhaitez utiliser des expressions conditionnelles et que vous ne

pouvez pas utiliser la MRSC pour une raison quelconque, ou lorsque vous devez effectuer des transactions. Ces expressions ne sont possibles que si vous savez que vous agissez sur la base des données les plus récentes. Elles nécessitent donc d'envoyer toutes les demandes d'écriture vers une seule région disposant des données les plus récentes.

Lorsque vous utilisez une table MRSC, vous pouvez choisir d'écrire généralement vers une région pour des raisons de commodité. Par exemple, cela peut aider à minimiser le développement de votre infrastructure au-delà de DynamoDB. Le mode d'écriture resterait l'écriture dans n'importe quelle région, car avec la MRSC, vous pouvez écrire en toute sécurité dans n'importe quelle région à tout moment sans vous soucier de la résolution des conflits, ce qui obligerait les tables MREC à choisir d'écrire dans une région.

Les lectures éventuellement cohérentes peuvent aller vers n'importe quelle région de réplica pour réduire les latences. Les lectures fortement cohérentes doivent aller dans la seule région principale.



Il est parfois nécessaire de modifier la région active en réponse à une défaillance régionale. Certains utilisateurs modifient régulièrement la région actuellement active, par exemple en mettant en œuvre

un follow-the-sun déploiement. Cela place la région active à proximité de la zone géographique la plus active (généralement là où il fait jour, d'où son nom), ce qui se traduit par les opérations de lecture et d'écriture avec le plus faible temps de latence. Cela présente également l'avantage d'appeler quotidiennement le code de la région qui change, afin de s'assurer qu'il est bien testé avant toute reprise après sinistre.

La ou les régions passives peuvent conserver un ensemble réduit d'infrastructures autour de DynamoDB, qui ne se développe que si la région devient active. Ce guide ne couvre pas les modèles veilleuse et secours semi-automatique. Pour plus d'informations, voir Architecture de reprise après sinistre (DR) activée AWS, partie III : veilleuse et mode veille.

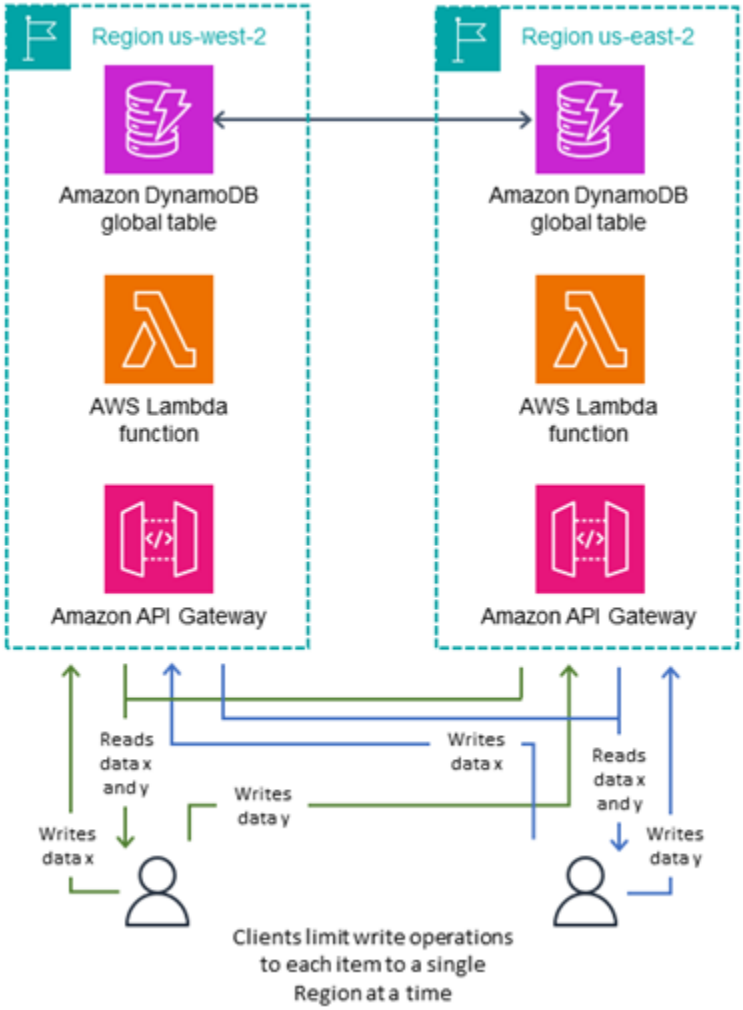
L'utilisation du mode Écrire dans une région fonctionne bien lorsque vous utilisez des tables globales pour des opérations de lecture distribuées à l'échelle mondiale à faible latence. Prenons l'exemple d'une grande entreprise de médias sociaux qui doit disposer des mêmes données de référence dans toutes les régions du monde. Elle ne met pas souvent à jour les données, mais lorsqu'elle le fait, elle n'écrit que dans une seule région afin d'éviter tout conflit d'écriture potentiel. Les opérations de lecture sont toujours autorisées dans toutes les régions.

Autre exemple : le client du secteur des services financiers dont nous avons parlé précédemment, et qui a mis en œuvre le calcul de remise en argent quotidien. Il a utilisé le mode Écrire dans n'importe quelle région pour calculer le solde, mais aussi le mode Écrire dans une région pour suivre les paiements. Ce travail nécessite des transactions, qui ne sont pas prises en charge dans les tables MRSC. Il fonctionne donc mieux avec une table MREC séparée et le mode Écrire dans une région.

Écrire dans votre région (primaire mixte)

Le mode d'écriture Écrire dans votre région, illustré dans le schéma suivant, fonctionne avec les tables MREC. Il attribue différents sous-ensembles de données aux différentes régions d'origine et autorise les opérations d'écriture sur un élément uniquement via sa région d'origine. Ce mode est actif-passif, mais assigne la région active en fonction de l'élément. Chaque région est primaire pour son propre jeu de données qui ne se chevauche pas et les opérations d'écriture doivent être protégées pour garantir une localisation correcte.

Ce mode est similaire à Écrire dans une région, sauf qu'il permet des opérations d'écriture à faible latence, car les données associées à chaque utilisateur peuvent être placées dans un réseau plus près de cet utilisateur. Cela permet également de répartir l'infrastructure environnante de manière plus uniforme entre les régions et de réduire le travail de construction de l'infrastructure lors d'un scénario de basculement, car une partie de l'infrastructure de toutes les régions est déjà active.



Vous pouvez déterminer la région d'origine des objets de plusieurs manières :

- Intrinsèque : certains aspects des données, tels qu'un attribut spécial ou une valeur vectorisée dans leur clé de partition, indiquent clairement leur région d'origine. Cette technique est décrite dans l'article de blog Utiliser l'épinglage des régions pour définir une région d'origine pour les éléments d'une table globale Amazon DynamoDB.
- Négocié : la région d'origine de chaque jeu de données est négociée de manière externe, par exemple avec un service global distinct qui gère les attributions. La mission peut avoir une durée limitée, après laquelle elle peut faire l'objet d'une renégociation.
- Orienté vers les tables : au lieu de créer une seule table globale de réplication, vous créez le même nombre de tables globales que de régions de réplication. Le nom de chaque table indique sa région d'origine. Dans les opérations standard, toutes les données sont écrites dans la région d'origine tandis que les autres régions conservent une copie en lecture seule. Lors d'un basculement, une autre région adopte temporairement des fonctions d'écriture pour cette table.

Imaginons que vous travaillez pour une entreprise de jeux vidéo. Vous avez besoin d'opérations de lecture et d'écriture à faible latence pour tous les joueurs du monde entier. Vous attribuez à chaque joueur la région la plus proche de lui. Cette région prend en charge toutes ses opérations de lecture et d'écriture, garantissant ainsi une forte read-after-write cohérence. Toutefois, lorsqu'un joueur voyage ou si sa région d'origine est en panne, une copie complète de ses données est disponible dans d'autres régions. Le joueur peut alors être attribué à une autre région d'origine.

Autre exemple, imaginez que vous travaillez pour une entreprise de visioconférence. À chaque téléconférence, des métadonnées sont attribuées à une région particulière. Les participants peuvent utiliser la région la plus proche d'eux pour minimiser la latence. En cas de panne d'une région, les tables globales permettent une restauration rapide, car le système peut déplacer le traitement de l'appel vers une autre région où il existe déjà une copie répliquée des données.

Récapitulatif

- Le mode Écrire dans n'importe quelle région convient aux tables MRSC et aux appels idempotents vers les tables MREC.
- Le mode Écrire dans une région convient aux appels non idempotents vers les tables MREC.
- Le mode Écrire dans votre région convient aux appels non idempotents vers les tables MREC, où il est important que les clients écrivent dans une région proche d'eux.

Stratégies de routage dans DynamoDB

La partie la plus complexe d'un déploiement de tables globales est peut-être la gestion du routage des demandes. Les demandes doivent d'abord aller d'un utilisateur final vers une région choisie et acheminée d'une manière ou d'une autre. La demande rencontre une pile de services dans cette région, notamment une couche de calcul qui consiste peut-être en un équilibreur de charge soutenu par une AWS Lambda fonction, un conteneur ou un nœud Amazon Elastic Compute Cloud (Amazon EC2), et éventuellement d'autres services, notamment une autre base de données. Cette couche de calcul communique avec DynamoDB. Pour ce faire, elle doit utiliser le point de terminaison local de cette région. Les données de la table globale sont répliquées dans toutes les autres régions participantes et chaque région dispose d'une pile de services similaire autour de sa table DynamoDB.

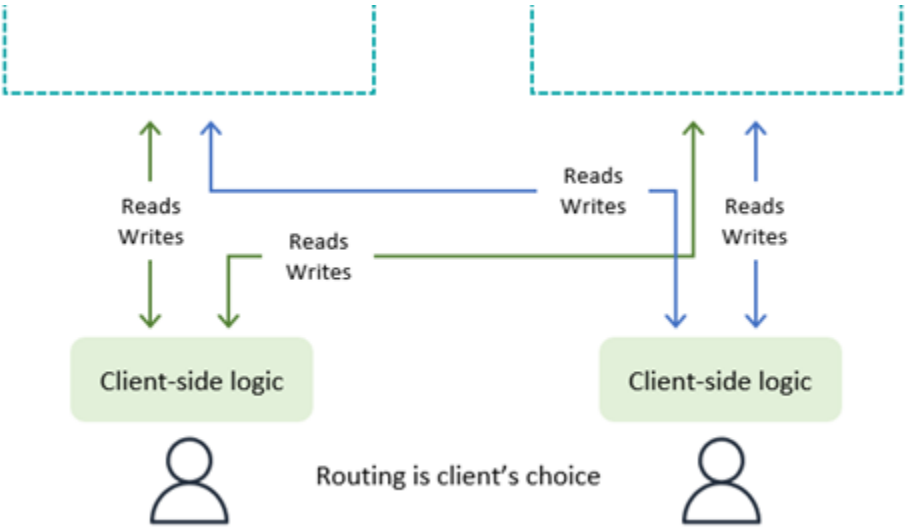
La table globale fournit à chaque pile des différentes régions une copie locale des mêmes données. Vous pouvez envisager de concevoir une pile unique dans une seule région et prévoir de passer des appels distants vers le point de terminaison DynamoDB d'une région secondaire en cas de problème avec la table DynamoDB locale. Ce n'est pas la bonne pratique recommandée. Si un problème dans

une région est dû à DynamoDB (ou, plus probablement, à un autre élément de la pile ou à un autre service dépendant de DynamoDB), il est préférable d'acheminer l'utilisateur final vers une autre région pour le traitement et d'utiliser la couche de calcul de cette autre région, qui communiquera avec son point de terminaison DynamoDB local. Cette approche permet de contourner entièrement la région problématique. Pour garantir la résilience, vous avez besoin d'une réplication entre plusieurs régions, avec une réplication de la couche de calcul ainsi que de la couche de données.

Il existe de nombreuses techniques alternatives pour acheminer une demande d'utilisateur final vers une région à des fins de traitement. Le choix optimal dépend de votre mode d'écriture et de vos considérations relatives au basculement. Cette section décrit quatre options : piloté par le client, couche de calcul, Route 53 et Global Accelerator.

Routage des demandes piloté par le client

Avec le routage des demandes piloté par le client, illustré dans le schéma suivant, le client de l'utilisateur final (une application, une page Web ou un autre client) garde une trace des points de terminaison d'application valides (par exemple, un point de terminaison Amazon API Gateway plutôt qu'un point de terminaison DynamoDB littéral) et utilise sa propre logique intégrée pour choisir la région avec JavaScript laquelle communiquer. Il peut choisir de façon aléatoire, en fonction des latences les plus faibles observées, des mesures de bande passante les plus élevées observées ou des surveillances de l'état effectuées localement.



L'avantage du routage des demandes piloté par le client est qu'il peut s'adapter à des facteurs tels que les conditions réelles du trafic Internet public pour changer de région en cas de dégradation des performances. Le client doit connaître tous les points de terminaison potentiels, mais il n'est pas courant de lancer un nouveau point de terminaison régional.

Avec le mode Écrire dans n'importe quelle région, un client peut sélectionner unilatéralement son point de terminaison préféré. Si son accès à une région est perturbé, le client peut effectuer un routage vers un autre point de terminaison.

Avec le mode Écrire dans une région, le client aura besoin d'un mécanisme pour acheminer ses écritures vers la région actuellement active. Cela peut être aussi simple que de tester de façon empirique quelle région accepte actuellement les écritures (en notant tout rejet d'écriture et en revenant à une autre région) ou aussi complexe que d'appeler un coordinateur mondial pour demander l'état actuel de l'application (ce qui peut-être basé sur le contrôle de routage Amazon Application Recovery Controller (ARC) qui fournit un système piloté par un quorum à 5 régions afin de maintenir l'état global pour des besoins tels que celui-ci). Le client peut décider si les lectures peuvent être acheminées vers n'importe quelle région pour une cohérence éventuelle ou si elles doivent être acheminées vers la région active pour une forte cohérence. Pour plus d'informations, consultez Fonctionnement de Route 53.

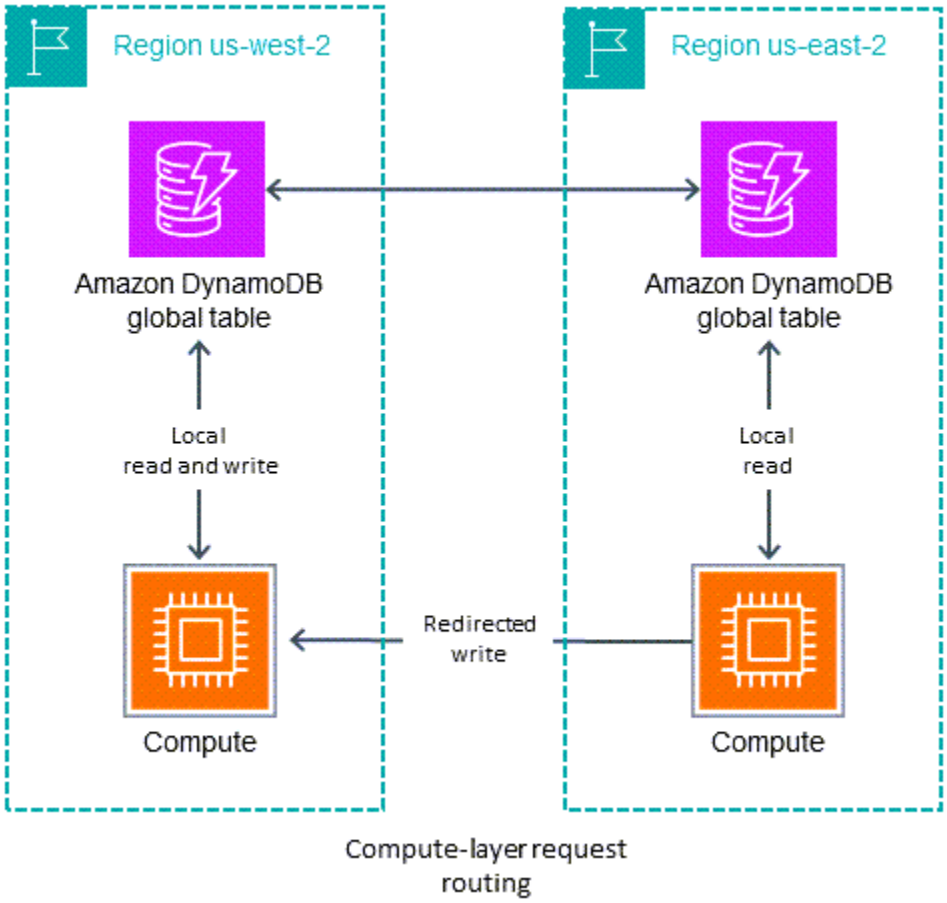
Avec le mode Écrire dans votre région, le client doit déterminer la région d'origine du jeu de données sur lequel il travaille. Par exemple, si le client correspond à un compte utilisateur et que chaque compte utilisateur est associé à une région, le client peut demander le point de terminaison approprié auprès d'un système de connexion global.

Par exemple, une entreprise de services financiers qui aide les utilisateurs à gérer les finances de leur entreprise via le Web peut utiliser des tables globales avec un mode Écrire dans votre région. Chaque utilisateur doit se connecter à un service central. Ce service renvoie les informations d'identification et le point de terminaison de la région dans laquelle ces informations d'identification fonctionnent. Les informations d'identification sont valides pendant une courte période. Ensuite, la page Web négocie automatiquement une nouvelle connexion, ce qui permet de rediriger potentiellement l'activité de l'utilisateur vers une nouvelle région.

Routage des demandes dans la couche de calcul

Avec le routage des demandes dans la couche de calcul, illustré dans le diagramme suivant, le code qui s'exécute dans la couche de calcul décide s'il souhaite traiter la demande localement ou la transmettre à une copie de lui-même qui s'exécute dans une autre région. Lorsque vous utilisez le mode Écrire dans une région, la couche de calcul peut détecter qu'il ne s'agit pas de la région active et autoriser les opérations de lecture locales tout en transférant toutes les opérations d'écriture vers une autre région. Ce code de couche de calcul doit tenir compte de la topologie des données et des règles de routage, et les appliquer de manière fiable en fonction des derniers paramètres qui spécifient quelles régions sont actives pour quelles données. La pile logicielle externe au sein de

la région n'a pas besoin de savoir comment les demandes de lecture et d'écriture sont acheminées par le microservice. Dans une conception robuste, la région réceptrice vérifie s'il s'agit de la région principale actuelle pour l'opération d'écriture. Si ce n'est pas le cas, cela génère une erreur indiquant que l'état global doit être corrigé. La région réceptrice peut également mettre en mémoire tampon l'opération d'écriture pendant un certain temps si la région principale est en cours de modification. Dans tous les cas, la pile de calcul d'une région écrit uniquement sur son point de terminaison DynamoDB local, mais les piles de calcul peuvent communiquer entre elles.

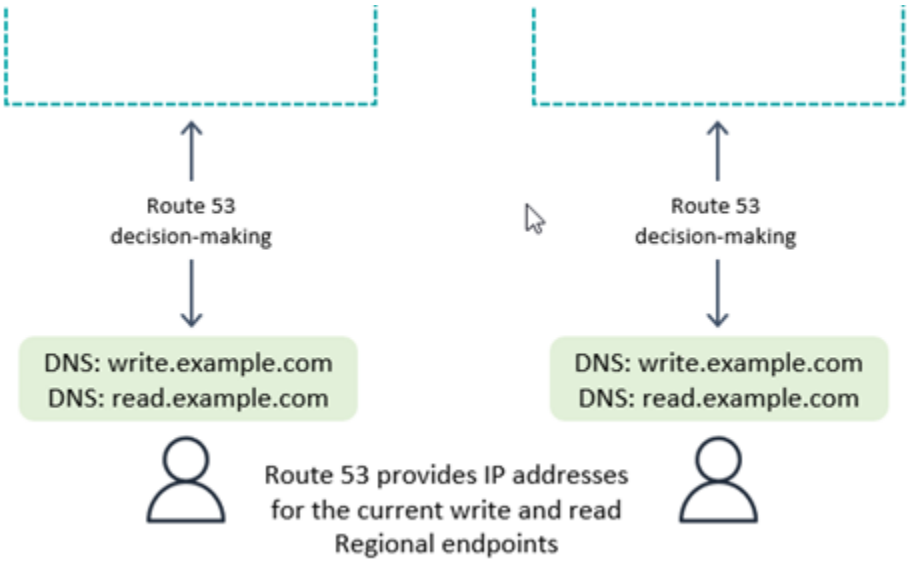


Le Vanguard Group utilise un système appelé Global Orchestration and Status Tool (GOaST) et une bibliothèque appelée Global Multi-Region library (GMRlib) pour ce processus de routage, tel que présenté à re:Invent 2022. Ils utilisent un follow-the-sun seul modèle principal. GOaST conserve l'état global, de la même manière que le contrôle de routage ARC décrit dans la section précédente. Il utilise une table globale pour savoir quelle région est la région principale et quand le prochain commutateur principal est planifié. Toutes les opérations de lecture et d'écriture sont effectuées GMRlib, ce qui est coordonné avec GOa ST. GMRlib permet d'effectuer des opérations de lecture localement, avec une faible latence. Pour les opérations d'écriture, GMRlib vérifie si la région locale est la région principale actuelle. Si tel est le cas, l'opération d'écriture se termine directement. Dans

le cas contraire, GMRlib transmet la tâche d'écriture GMRlib à la région principale. Cette bibliothèque réceptrice confirme qu'elle se considère également comme la région principale et génère une erreur si ce n'est pas le cas, ce qui indique un délai de propagation par rapport à l'état global. Cette approche offre un avantage en matière de validation car elle ne permet pas d'écrire directement sur un point de terminaison DynamoDB distant.

Routage des demandes avec Route 53

Amazon Application Recovery Controller (ARC) est une technologie de service de nom de domaine (DNS). Avec Route 53, le client demande son point de terminaison en recherchant un nom de domaine DNS connu et Route 53 renvoie l'adresse IP correspondante au ou aux points de terminaison régionaux qu'il juge les plus appropriés. Le diagramme suivant en est l'illustration. Route 53 dispose d'une longue liste de politiques de routage qu'elle utilise pour déterminer la région appropriée. Il peut également effectuer un routage de basculement pour acheminer le trafic en dehors des régions où les surveillances de l'état échouent.



Avec le mode Écrire dans n'importe quelle région, ou en combinaison avec le routage des demandes dans la couche de calcul sur le back-end, Route 53 peut bénéficier d'un accès complet pour renvoyer la région en fonction de règles internes complexes, telles que la région la plus proche du réseau, la zone géographique la plus proche, ou tout autre choix.

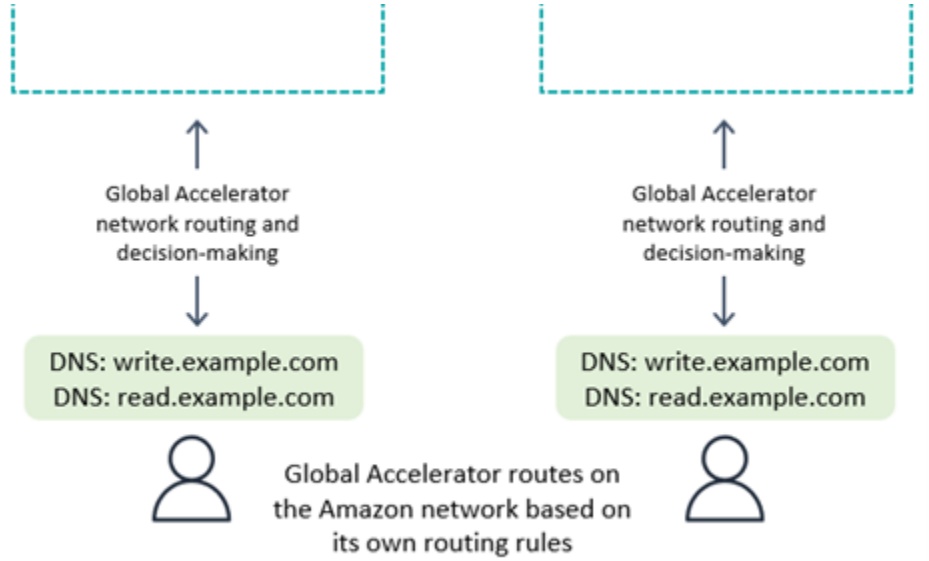
Avec le mode Écrire dans une région, vous pouvez configurer Route 53 pour renvoyer la région actuellement active (à l'aide de Route 53 ARC). Si le client souhaite se connecter à une région passive (par exemple, pour des opérations de lecture), il peut rechercher un autre nom DNS.

> **Note**
>
> Les clients mettent en cache les adresses IP figurant dans la réponse de Route 53 pendant une durée indiquée par le paramètre de durée de vie (TTL) du nom de domaine. Une durée de vie plus longue prolonge l'objectif de délai de reprise (RTO) pour que tous les clients puissent reconnaître le nouveau point de terminaison. Une valeur de 60 secondes est généralement utilisée pour un basculement. Tous les logiciels ne respectent pas parfaitement l'expiration du TTL DNS, et il peut y avoir plusieurs niveaux de mise en cache DNS, par exemple au niveau du système d'exploitation, de la machine virtuelle et de l'application.

Avec le mode Écrire dans votre région, il est préférable d'éviter Route 53, sauf si vous utilisez également le routage des demandes dans la couche de calcul.

Routage des demandes avec Global Accelerator

Avec AWS Global Accelerator, comme illustré dans le diagramme suivant, un client utilise Route 53 pour rechercher le nom de domaine connu. Cependant, au lieu de récupérer une adresse IP correspondant à un point de terminaison régional, le client reçoit une adresse IP statique anycast qui achemine vers l'emplacement AWS périphérique le plus proche. À partir de cet emplacement périphérique, tout le trafic est acheminé sur le réseau AWS privé et vers un point de terminaison (tel qu'un équilibreur de charge ou API Gateway) dans une région choisie par des règles de routage qui sont gérées dans Global Accelerator. Comparé au routage basé sur les règles Route 53, le routage des demandes avec Global Accelerator présente des latences plus faibles car il réduit le volume de trafic sur l'Internet public. De plus, comme Global Accelerator ne dépend pas de l'expiration de la durée de vie du DNS pour modifier les règles de routage, il peut ajuster le routage plus rapidement.



Avec le mode Écrire dans n'importe quelle région, ou s'il est associé au routage des demandes dans la couche de calcul sur le back-end, Global Accelerator fonctionne parfaitement. Le client se connecte à l'emplacement périphérique le plus proche et n'a pas à se soucier de savoir quelle région reçoit la demande.

Avec le mode Écrire dans une région, les règles de routage de Global Accelerator doivent envoyer des demandes à la région actuellement active. Vous pouvez utiliser des surveillances de l'état qui signalent artificiellement une défaillance dans une région qui n'est pas considérée par votre système global comme étant la région active. Comme avec DNS, il est possible d'utiliser un autre nom de domaine DNS pour acheminer les demandes de lecture si celles-ci peuvent provenir de n'importe quelle région.

Avec le mode Écrire dans votre région, il est préférable d'éviter Global Accelerator, sauf si vous utilisez également le routage des demandes dans la couche de calcul.

Processus d'évacuation

L'évacuation d'une région est le processus d'activité de migration, généralement des activités de lecture et d'écriture ou des activités d'écriture, hors de cette région.

Évacuation d'une région active

Vous pouvez décider d'évacuer une région active pour plusieurs raisons : dans le cadre de vos activités commerciales habituelles (par exemple, si vous utilisez le mode « écrire dans une région ») follow-the-sun, en raison de la décision commerciale de modifier la région actuellement active,

en réponse à des défaillances de la pile logicielle en dehors de DynamoDB, ou parce que vous rencontrez des problèmes généraux tels que des latences plus élevées que d'habitude au sein de la région.

Avec le mode Écrire dans n'importe quelle région, il est facile d'évacuer une région active. Vous pouvez acheminer le trafic vers d'autres régions via n'importe quel système de routage et laisser les opérations d'écriture dans la région évacuée se répliquer comme d'habitude.

Les modes Écrire dans une région et Écrire dans votre région sont généralement utilisés avec les tables MREC. Par conséquent, vous devez vous assurer que toutes les opérations d'écriture dans la région active ont été entièrement enregistrées, traitées en flux et propagées globalement avant de commencer les opérations d'écriture dans la nouvelle région active, afin de garantir que les futures opérations d'écriture seront traitées par rapport à la dernière version des données.

Supposons que la région A soit active et que la région B soit passive (soit pour la table complète, soit pour les éléments qui se trouvent dans la région A). Le mécanisme habituel pour une évacuation consiste à suspendre les opérations d'écriture vers A, à attendre suffisamment longtemps pour que ces opérations se propagent entièrement vers B, à mettre à jour la pile d'architecture pour reconnaître B comme étant active, puis à reprendre les opérations d'écriture vers B. Aucune métrique n'indique avec une certitude absolue que la région A a entièrement répliqué ses données vers la région B. Si la région A est saine, suspendre les opérations d'écriture dans la région A et attendre 10 fois la valeur maximale récente de la métrique `ReplicationLatency` serait généralement suffisant pour déterminer si la réplication est terminée. Si la région A n'est pas saine et indique d'autres zones présentant des latences accrues, vous devez choisir un multiple plus élevé pour le temps d'attente.

Évacuation d'une région déconnectée

Il y a un cas particulier à prendre en compte : et si la région A se déconnectait complètement sans préavis ? C'est un cas très peu probable, mais il doit néanmoins être pris en compte.

Évacuation d'une table MRSC hors ligne

> Si cela se produit avec une table MRSC, vous n'avez rien de spécial à faire. Les tables MRSC prennent en charge un objectif de point de reprise (RPO) de zéro. Toutes les opérations d'écriture réussies effectuées sur la table MRSC dans la région hors ligne seront disponibles dans toutes les autres tables de régions. Il n'y a donc aucune lacune potentielle dans les données, même si la région se déconnecte complètement sans préavis. Les entreprises peuvent continuer à utiliser des réplicas situés dans les autres régions.

Évacuation d’une table MREC hors ligne

Si cela se produit avec une table MREC, toutes les opérations d’écriture dans la région A qui n’ont pas encore été propagées sont conservées et propagées après la remise en ligne de la région A. Les opérations d’écriture ne sont pas perdues, mais leur propagation est retardée indéfiniment.

La procédure à suivre dans ce cas dépend de la décision de l’application. Pour la continuité des activités, les opérations d’écriture peuvent devoir être effectuées vers la nouvelle région principale B. Toutefois, si un élément de la région B reçoit une mise à jour alors qu’une opération d’écriture pour cet élément est en attente de propagation depuis la région A, la propagation est supprimée selon le modèle victoire du dernier auteur. Toute mise à jour dans la région B peut supprimer une demande d’écriture entrante.

Avec le mode Écrire dans n’importe quelle région, les opérations de lecture et d’écriture peuvent se poursuivre dans la région B, en sachant que les éléments de la région A finiront par se propager dans la région B et en reconnaissant la possibilité que des éléments soient manquants jusqu’à ce que la région A revienne en ligne. Dans la mesure du possible, comme avec les opérations d’écriture idempotentes, vous devez envisager de rejouer le trafic d’écriture récent (par exemple, en utilisant une source d’événements en amont) afin de combler les lacunes liées aux opérations d’écriture potentiellement manquantes et de laisser la résolution du conflit de la victoire du dernier auteur supprimer la propagation éventuelle de l’opération d’écriture entrante.

Avec les autres modes d'écriture, vous devez considérer dans quelle mesure le travail peut se poursuivre avec une légère out-of-date vue du monde. Certaines opérations d’écriture de courte durée, telles que suivies par `ReplicationLatency`, sont absentes jusqu’à ce que la région A revienne en ligne. Les entreprises peuvent-elles aller de l’avant ? Dans certains cas d’utilisation, c’est possible, mais dans d’autres, pas sans des mécanismes d’atténuation supplémentaires.

Imaginons par exemple que vous deviez maintenir un solde créditeur disponible sans interruption, même en cas de défaillance complète d’une région. Vous pouvez diviser le solde en deux éléments différents, l’un réservé à la région A et l’autre à la région B, et commencer chacun avec la moitié du solde disponible. Cela utiliserait le mode Écrire dans votre région. Les mises à jour transactionnelles traitées dans chaque région seraient comparées à la copie locale du solde. Si la région A est complètement déconnectée, le traitement des transactions pourrait toujours se poursuivre dans la région B et les opérations d’écriture seraient limitées à la partie du solde détenue dans la région B. Le fractionnement du solde introduit des difficultés lorsque le solde baisse ou que le crédit doit être rééquilibré, mais cela fournit un exemple de reprise de l’entreprise sûre, même en cas d’opérations d’écriture incertaines en cours.

Autre exemple, imaginez que vous capturez des données de formulaire Web. Vous pouvez utiliser Optimistic Concurrency Control (OCC) pour attribuer des versions aux éléments de données et intégrer la dernière version dans le formulaire Web en tant que champ masqué. À chaque envoi, l'opération d'écriture ne réussit que si la version de la base de données correspond toujours à la version avec laquelle le formulaire a été créé. Si les versions ne correspondent pas, le formulaire Web peut être actualisé (ou soigneusement fusionné) avec la version actuelle de la base de données et l'utilisateur peut recommencer. Le modèle OCC offre généralement une protection contre l'écrasement par un autre client et la production d'une nouvelle version des données, mais il peut également être utile en cas de basculement lorsqu'un client peut rencontrer d'anciennes versions des données. Imaginons que vous utilisez l'horodatage comme version. Le formulaire A été créé pour la première fois pour la région A à midi mais qu'il essaie (après le basculement) d'écrire dans la région B et remarque que la dernière version de la base de données est 11 h 59. Dans ce scénario, le client peut soit attendre que la version de 12 h 00 se propage vers la région B, puis écrire sur cette version, soit utiliser la version de 11 h 59 et créer une nouvelle version à 12 h 01 (qui, après écriture, supprime la version entrante une fois la région A rétablie).

Troisième exemple, une entreprise de services financiers conserve les données relatives aux comptes clients et à leurs transactions financières dans une base de données DynamoDB. En cas de panne complète de la région A, elle veut s'assurer que toute activité d'écriture liée à ses comptes est entièrement disponible dans la région B, ou elle souhaite mettre ses comptes en quarantaine et les considérer comme partiels jusqu'à ce que la région A revienne en ligne. Au lieu de suspendre toutes les activités, elle a décidé de suspendre uniquement celles de l'infime fraction des comptes qui, selon elle, contenaient des transactions non propagées. Pour ce faire, elle a utilisé une troisième région, que nous appellerons région C. Avant de traiter les opérations d'écriture dans la région A, elle a placé un résumé succinct des opérations en attente (par exemple, un nouveau nombre de transactions pour un compte) dans la région C. Ce résumé était suffisant pour permettre à la région B de déterminer si sa vue était entièrement à jour. Cette action a effectivement verrouillé le compte entre le moment de l'écriture dans la région C et le moment où la région A a accepté les opérations d'écriture et que la région B les a reçues. Les données de la région C n'ont pas été utilisées, sauf dans le cadre d'un processus de basculement, après quoi la région B a pu recouper ses données avec la région C pour vérifier si l'un de ses comptes était obsolète. Ces comptes seraient marqués comme mis en quarantaine jusqu'à ce que la restauration de la région A propage les données partielles vers la région B. En cas de défaillance de la région C, une nouvelle région D pourrait être utilisée à la place. Les données de la région C étaient très transitoires et, au bout de quelques minutes, la région D disposerait d'un up-to-date enregistrement suffisant des opérations d'écriture en vol pour être pleinement utile. En cas d'échec de la région B, la région A pourrait continuer à accepter les demandes d'écriture en

coopération avec la région C. Cette entreprise était prête à accepter des écritures à latence plus élevée (vers deux régions : C puis A) et a eu la chance de disposer d'un modèle de données permettant de résumer succinctement l'état d'un compte.

Planification de la capacité de débit pour les tables globales DynamoDB

La migration du trafic d'une région vers une autre nécessite un examen attentif des paramètres de la table DynamoDB concernant la capacité.

Voici quelques considérations relatives à la gestion de la capacité d'écriture :

- Une table globale doit être en mode à la demande ou provisionné avec Auto Scaling activé.
- En cas de provisionnement avec Auto Scaling, les paramètres d'écriture (utilisation minimale, maximale et cible) sont répliqués entre les régions. Bien que les paramètres d'autoscaling soient synchronisés, la capacité d'écriture réellement provisionnée peut varier indépendamment entre les régions.
- L'une des raisons pour lesquelles vous pouvez constater une capacité d'écriture allouée différente est due à la fonctionnalité de durée de vie. Lorsque vous activez la durée de vie dans DynamoDB, vous pouvez spécifier un nom d'attribut dont la valeur indique l'heure d'expiration de l'élément, au format d'époque Unix, en secondes. Passé ce délai, DynamoDB peut supprimer l'élément sans frais d'écriture. Avec les tables globales, vous configurez la durée de vie dans n'importe quelle région et ce paramètre est répliqué automatiquement dans les autres régions associées à la table globale. Lorsqu'un élément peut être supprimé par le biais d'une règle de durée de vie, ce travail peut être effectué dans n'importe quelle région. L'opération de suppression est effectuée sans consommer d'unités d'écriture sur la table source, mais les tables de réplicas obtiennent une écriture répliquée de cette opération de suppression et entraînent des coûts unitaires d'écriture répliqués. La TTL n'est pas prise en charge dans les tables MRSC.
- Si vous utilisez Auto Scaling, assurez-vous que le paramètre de capacité d'écriture maximale provisionnée est suffisamment élevé pour gérer toutes les opérations d'écriture ainsi que toutes les opérations de suppression de la durée de vie potentielles. Auto Scaling ajuste chaque région en fonction de sa consommation d'écriture. Les tables à la demande n'ont pas de paramètre de capacité d'écriture maximale provisionnée, mais la limite de débit d'écriture maximal au niveau de la table indique la capacité d'écriture soutenue maximale autorisée par la table à la demande. La limite par défaut est de 40 000, mais elle peut être ajustée. Nous vous recommandons de la définir à un niveau suffisamment élevé pour gérer toutes les opérations d'écriture (y compris les opérations d'écriture de la durée de vie) dont la table à la demande peut avoir besoin. Cette

valeur doit être la même dans toutes les régions participantes lorsque vous configurez des tables globales.

Voici quelques considérations relatives à la gestion de la capacité de lecture :

- Les paramètres de gestion de la capacité de lecture peuvent différer entre les régions car il est supposé que les différentes régions peuvent avoir des modèles de lecture indépendants. La première fois que vous ajoutez un réplica global à une table, la capacité de la région source est propagée. Après sa création, vous pouvez ajuster les paramètres de capacité de lecture, qui ne sont pas transférés de l'autre côté.
- Lorsque vous utilisez Auto Scaling de DynamoDB, veillez à ce que les paramètres de capacité de lecture maximale allouée soient suffisamment élevés pour gérer toutes les opérations de lecture dans toutes les régions. Au cours des opérations standard, la capacité de lecture peut être répartie entre les régions, mais lors du basculement, la table doit pouvoir s'adapter automatiquement à l'augmentation de la charge de travail de lecture. Les tables à la demande n'ont pas de paramètre de capacité de lecture maximale provisionnée, mais la limite de débit de lecture maximal au niveau de la table indique la capacité de lecture soutenue maximale autorisée par la table à la demande. La limite par défaut est de 40 000, mais elle peut être ajustée. Nous vous recommandons de la définir à un niveau suffisamment élevé pour gérer toutes les opérations de lecture dont la table pourrait avoir besoin si toutes les opérations de lecture devaient être acheminées vers cette région unique.
- Si une table d'une région ne reçoit généralement pas de trafic de lecture mais qu'elle doit en absorber une grande partie après un basculement, vous pouvez préchauffer la capacité de la table pour qu'elle accepte un niveau de trafic de lecture supérieur.

ARC propose des contrôles de préparation qui peuvent être utiles pour confirmer que les régions DynamoDB possèdent des paramètres de table et des quotas de comptes similaires, que vous utilisiez Route 53 ou non pour acheminer les demandes. Ces contrôles de préparation peuvent également aider à ajuster les quotas au niveau du compte pour s'assurer qu'ils correspondent.

Liste de vérification pour la préparation des tables globales DynamoDB

Utilisez la liste de contrôle suivante pour les décisions et les tâches lorsque vous déployez des tables globales.

- Déterminez quel mode de cohérence serait plus bénéfique pour votre cas d'utilisation parmi les modes MRSC et MREC. Avez-vous besoin d'une cohérence élevée, même avec une latence supérieure et d'autres compromis ?
- Déterminez combien et quelles régions doivent participer à la table globale. Si vous envisagez d'utiliser MRSC, déterminez si vous voulez que la troisième région soit un réplica ou un témoin.
- Déterminez le mode d'écriture de votre application. Il est différent du mode de cohérence. Pour de plus amples informations, veuillez consulter Modes d'écriture avec des tables globales DynamoDB.
- Planifiez votre stratégie Stratégies de routage dans DynamoDB en fonction de votre mode d'écriture.
- Définissez vos the section called "Processus d'évacuation"

L'évacuation d'une région est le processus d'activité de migration, généralement des activités de lecture et d'écriture ou des activités d'écriture, hors de cette région.

Évacuation d'une région active

Vous pouvez décider d'évacuer une région active pour plusieurs raisons : dans le cadre de vos activités commerciales habituelles (par exemple, si vous utilisez le mode « écrire dans une région ») follow-the-sun, en raison de la décision commerciale de modifier la région actuellement active, en réponse à des défaillances de la pile logicielle en dehors de DynamoDB, ou parce que vous rencontrez des problèmes généraux tels que des latences plus élevées que d'habitude au sein de la région.

Avec le mode Écrire dans n'importe quelle région, il est facile d'évacuer une région active. Vous pouvez acheminer le trafic vers d'autres régions via n'importe quel système de routage et laisser les opérations d'écriture dans la région évacuée se répliquer comme d'habitude.

Les modes Écrire dans une région et Écrire dans votre région sont généralement utilisés avec les tables MREC. Par conséquent, vous devez vous assurer que toutes les opérations d'écriture dans la région active ont été entièrement enregistrées, traitées en flux et propagées globalement avant de commencer les opérations d'écriture dans la nouvelle région active, afin de garantir que les futures opérations d'écriture seront traitées par rapport à la dernière version des données.

Supposons que la région A soit active et que la région B soit passive (soit pour la table complète, soit pour les éléments qui se trouvent dans la région A). Le mécanisme habituel pour une évacuation consiste à suspendre les opérations d'écriture vers A, à attendre suffisamment longtemps pour que ces opérations se propagent entièrement vers B, à mettre à jour la pile d'architecture pour reconnaître B comme étant active, puis à reprendre les opérations d'écriture

vers B. Aucune métrique n'indique avec une certitude absolue que la région A a entièrement répliqué ses données vers la région B. Si la région A est saine, suspendre les opérations d'écriture dans la région A et attendre 10 fois la valeur maximale récente de la métrique `ReplicationLatency` serait généralement suffisant pour déterminer si la réplication est terminée. Si la région A n'est pas saine et indique d'autres zones présentant des latences accrues, vous devez choisir un multiple plus élevé pour le temps d'attente.

Évacuation d'une région déconnectée

Il y a un cas particulier à prendre en compte : et si la région A se déconnectait complètement sans préavis ? C'est un cas très peu probable, mais il doit néanmoins être pris en compte.

Évacuation d'une table MRSC hors ligne

Si cela se produit avec une table MRSC, vous n'avez rien de spécial à faire. Les tables MRSC prennent en charge un objectif de point de reprise (RPO) de zéro. Toutes les opérations d'écriture réussies effectuées sur la table MRSC dans la région hors ligne seront disponibles dans toutes les autres tables de régions. Il n'y a donc aucune lacune potentielle dans les données, même si la région se déconnecte complètement sans préavis. Les entreprises peuvent continuer à utiliser des réplicas situés dans les autres régions.

Évacuation d'une table MREC hors ligne

Si cela se produit avec une table MREC, toutes les opérations d'écriture dans la région A qui n'ont pas encore été propagées sont conservées et propagées après la remise en ligne de la région A. Les opérations d'écriture ne sont pas perdues, mais leur propagation est retardée indéfiniment.

La procédure à suivre dans ce cas dépend de la décision de l'application. Pour la continuité des activités, les opérations d'écriture peuvent devoir être effectuées vers la nouvelle région principale B. Toutefois, si un élément de la région B reçoit une mise à jour alors qu'une opération d'écriture pour cet élément est en attente de propagation depuis la région A, la propagation est supprimée selon le modèle victoire du dernier auteur. Toute mise à jour dans la région B peut supprimer une demande d'écriture entrante.

Avec le mode Écrire dans n'importe quelle région, les opérations de lecture et d'écriture peuvent se poursuivre dans la région B, en sachant que les éléments de la région A finiront par se propager dans la région B et en reconnaissant la possibilité que des éléments soient manquants jusqu'à ce que la région A revienne en ligne. Dans la mesure du possible,

comme avec les opérations d'écriture idempotentes, vous devez envisager de rejouer le trafic d'écriture récent (par exemple, en utilisant une source d'événements en amont) afin de combler les lacunes liées aux opérations d'écriture potentiellement manquantes et de laisser la résolution du conflit de la victoire du dernier auteur supprimer la propagation éventuelle de l'opération d'écriture entrante.

Avec les autres modes d'écriture, vous devez considérer dans quelle mesure le travail peut se poursuivre avec une légère out-of-date vue du monde. Certaines opérations d'écriture de courte durée, telles que suivies par `ReplicationLatency`, sont absentes jusqu'à ce que la région A revienne en ligne. Les entreprises peuvent-elles aller de l'avant ? Dans certains cas d'utilisation, c'est possible, mais dans d'autres, pas sans des mécanismes d'atténuation supplémentaires.

Imaginons par exemple que vous deviez maintenir un solde créditeur disponible sans interruption, même en cas de défaillance complète d'une région. Vous pouvez diviser le solde en deux éléments différents, l'un réservé à la région A et l'autre à la région B, et commencer chacun avec la moitié du solde disponible. Cela utiliserait le mode Écrire dans votre région. Les mises à jour transactionnelles traitées dans chaque région seraient comparées à la copie locale du solde. Si la région A est complètement déconnectée, le traitement des transactions pourrait toujours se poursuivre dans la région B et les opérations d'écriture seraient limitées à la partie du solde détenue dans la région B. Le fractionnement du solde introduit des difficultés lorsque le solde baisse ou que le crédit doit être rééquilibré, mais cela fournit un exemple de reprise de l'entreprise sûre, même en cas d'opérations d'écriture incertaines en cours.

Autre exemple, imaginez que vous capturez des données de formulaire Web. Vous pouvez utiliser Optimistic Concurrency Control (OCC) pour attribuer des versions aux éléments de données et intégrer la dernière version dans le formulaire Web en tant que champ masqué. À chaque envoi, l'opération d'écriture ne réussit que si la version de la base de données correspond toujours à la version avec laquelle le formulaire a été créé. Si les versions ne correspondent pas, le formulaire Web peut être actualisé (ou soigneusement fusionné) avec la version actuelle de la base de données et l'utilisateur peut recommencer. Le modèle OCC offre généralement une protection contre l'écrasement par un autre client et la production d'une nouvelle version des données, mais il peut également être utile en cas de basculement lorsqu'un client peut rencontrer d'anciennes versions des données. Imaginons que vous utilisez l'horodatage comme version. Le formulaire A été créé pour la première fois pour la région A à midi mais qu'il essaie (après le basculement) d'écrire dans la région B et remarque

que la dernière version de la base de données est 11 h 59. Dans ce scénario, le client peut soit attendre que la version de 12 h 00 se propage vers la région B, puis écrire sur cette version, soit utiliser la version de 11 h 59 et créer une nouvelle version à 12 h 01 (qui, après écriture, supprime la version entrante une fois la région A rétablie).

Troisième exemple, une entreprise de services financiers conserve les données relatives aux comptes clients et à leurs transactions financières dans une base de données DynamoDB. En cas de panne complète de la région A, elle veut s'assurer que toute activité d'écriture liée à ses comptes est entièrement disponible dans la région B, ou elle souhaite mettre ses comptes en quarantaine et les considérer comme partiels jusqu'à ce que la région A revienne en ligne. Au lieu de suspendre toutes les activités, elle a décidé de suspendre uniquement celles de l'infime fraction des comptes qui, selon elle, contenaient des transactions non propagées. Pour ce faire, elle a utilisé une troisième région, que nous appellerons région C. Avant de traiter les opérations d'écriture dans la région A, elle a placé un résumé succinct des opérations en attente (par exemple, un nouveau nombre de transactions pour un compte) dans la région C. Ce résumé était suffisant pour permettre à la région B de déterminer si sa vue était entièrement à jour. Cette action a effectivement verrouillé le compte entre le moment de l'écriture dans la région C et le moment où la région A a accepté les opérations d'écriture et que la région B les a reçues. Les données de la région C n'ont pas été utilisées, sauf dans le cadre d'un processus de basculement, après quoi la région B a pu recouper ses données avec la région C pour vérifier si l'un de ses comptes était obsolète. Ces comptes seraient marqués comme mis en quarantaine jusqu'à ce que la restauration de la région A propage les données partielles vers la région B. En cas de défaillance de la région C, une nouvelle région D pourrait être utilisée à la place. Les données de la région C étaient très transitoires et, au bout de quelques minutes, la région D disposerait d'un up-to-date enregistrement suffisant des opérations d'écriture en vol pour être pleinement utile. En cas d'échec de la région B, la région A pourrait continuer à accepter les demandes d'écriture en coopération avec la région C. Cette entreprise était prête à accepter des écritures à latence plus élevée (vers deux régions : C puis A) et a eu la chance de disposer d'un modèle de données permettant de résumer succinctement l'état d'un compte.

, en fonction de votre mode de cohérence, de votre mode d'écriture et de votre stratégie de routage.

- Capturez des indicateurs sur l'état, la latence et les erreurs dans chaque région. Pour obtenir la liste des métriques DynamoDB, consultez AWS le billet de blog Monitoring Amazon DynamoDB for Operational Awareness pour obtenir la liste des métriques à observer. Vous devez également

utiliser des canaris synthétiques (demandes artificielles conçues pour détecter les pannes, du nom du canari de la mine de charbon), ainsi que l'observation en direct du trafic client. Les problèmes n'apparaîtront pas tous dans les métriques de DynamoDB.

- Si vous utilisez la MREC, définissez des alarmes en cas d'une augmentation soutenue dans `ReplicationLatency`. Une augmentation peut indiquer une mauvaise configuration accidentelle dans laquelle la table globale possède des paramètres d'écriture différents selon les régions, ce qui entraîne l'échec des demandes répliquées et une augmentation des latences. Cela pourrait également indiquer qu'il y a une perturbation régionale. Un bon exemple est de générer une alerte si la moyenne récente dépasse 180 000 millisecondes. Vous pouvez également surveiller si la `ReplicationLatency` passe en-dessous de 0, ce qui indique que la réplication est bloquée.
- Attribuez des paramètres de lecture et d'écriture maximaux suffisants pour chaque table globale.
- Identifiez à l'avance les raisons de l'évacuation d'une région. Si la décision implique un jugement humain, documentez toutes les considérations. Ce travail doit être effectué avec soin à l'avance, sans stress.
- Conservez un runbook pour chaque action qui doit avoir lieu lorsque vous évacuez une région. En général, très peu de travail est nécessaire pour les tables globales, mais le déplacement du reste de la pile peut s'avérer complexe.

> **Note**
>
> Avec des procédures de basculement, il est recommandé de se baser uniquement sur les opérations du plan de données et non sur celles du plan de contrôle, car certaines opérations du plan de contrôle peuvent être dégradées en cas de défaillance d'une région.

Pour plus d'informations, consultez le billet de AWS blog Créer des applications résilientes avec les tables globales Amazon DynamoDB : partie 4.

- Testez régulièrement tous les aspects du runbook, y compris les évacuations régionales. Un runbook non testé est un runbook peu fiable.
- Envisagez d'utiliser AWS Resilience Hub pour évaluer la résilience de l'ensemble de votre application (y compris les tables globales). Il fournit une vue complète du statut de résilience global de votre portefeuille d'applications via son tableau de bord.
- Envisagez d'utiliser les contrôles de préparation ARC pour évaluer la configuration actuelle de votre application et suivre tout écart par rapport aux bonnes pratiques.

- Lorsque vous écrivez une surveillance de l'état à utiliser avec Route 53 ou Global Accelerator, effectuez une série d'appels qui couvrent l'intégralité du flux de base de données. Si vous limitez votre vérification pour confirmer uniquement que le point de terminaison DynamoDB est actif, vous ne serez pas en mesure de couvrir de nombreux modes de défaillance Gestion des identités et des accès AWS tels que les erreurs de configuration (IAM), les problèmes de déploiement de code, les défaillances de la pile en dehors de DynamoDB, les latences de lecture ou d'écriture supérieures à la moyenne, etc.

Questions fréquemment posées sur le déploiement des tables globales

Quel est le coût des tables globales ?

- Le prix d'une opération d'écriture dans une table DynamoDB classique est exprimé en unités de capacité d'écriture WCUs (pour les tables provisionnées) ou en unités de demande d'écriture WRUs () pour les tables à la demande. Si vous écrivez un élément de 5 Ko, il implique des frais de 5 unités. Le prix d'une écriture dans une table globale est exprimé en unités de capacité d'écriture répliquées (rWCUs, pour les tables provisionnées) ou en unités de demande d'écriture répliquées (rWRUs, pour les tables à la demande). r WCUs et r WRUs ont le même prix que et. WGUs WRUs
- Les changements de rWCU et de rWRU sont facturés dans chaque région où l'élément est écrit directement ou par réplication. Des frais de transfert de données entre régions s'appliquent.
- L'écriture dans un index secondaire global (GSI) est considérée comme une opération d'écriture locale et utilise des unités d'écriture normales.
- Aucune capacité réservée n'est disponible pour r WCUs ou r pour le WRUs moment. L'achat de capacité réservée WCUs peut être avantageux pour les tables qui GSIs consomment des unités d'écriture.
- Lorsque vous ajoutez une nouvelle région à une table globale, DynamoDB démarre automatiquement la nouvelle région et vous facture comme s'il s'agissait d'une restauration de table, en fonction de la taille en Go de la table. Il facture également des frais de transfert de données entre régions.

Quelles sont les régions prises en charge par les tables globales ?

La version 2019.11.21 (actuelle) de Global Tables prend en charge toutes les tables MREC et les ensembles de régions suivants Régions AWS pour les tables MRSC :

- Groupe de régions États-Unis : USA Est (Virginie du Nord), USA Est (Ohio), USA Ouest (Oregon)

- Groupe de régions UE : Europe (Francfort), Europe (Irlande), Europe (Londres), Europe (Paris)
- Groupe de régions AP : Asie-Pacifique (Tokyo), Asie-Pacifique (Séoul) et Asie-Pacifique (Osaka)

Comment sont GSIs gérées les tables globales ?

Dans les Tables globales version 2019.11.21 (actuelle), lorsque vous créez un GSI dans une région, il est automatiquement créé dans les autres régions participantes et automatiquement rempli.

Comment arrêter la réplication d'une table globale ?

- Vous pouvez supprimer une table de réplica de la même manière qu'une autre table. La suppression de la table globale arrête la réplication vers cette région et supprime la copie de la table conservée dans cette région. Toutefois, vous ne pouvez pas arrêter la réplication tout en conservant des copies de la table en tant qu'entités indépendantes, ni suspendre la réplication.
- Une table MRSC doit être déployée dans exactement trois régions. Pour supprimer les réplicas, vous devez les supprimer entièrement, ainsi que le témoin, afin que la table MRSC devienne une table locale.

Comment DynamoDB Streams interagit-il avec les tables globales ?

- Chaque table globale produit un flux indépendant basé sur toutes ses opérations d'écriture, d'où qu'elles proviennent. Vous pouvez choisir de consommer ce flux DynamoDB dans une région ou dans toutes les régions (indépendamment). Si vous souhaitez traiter des opérations d'écritures locales mais pas répliquées, vous pouvez ajouter votre propre attribut de région à chaque élément afin d'identifier la région d'écriture. Vous pouvez ensuite utiliser un filtre d'événements Lambda pour appeler uniquement la fonction Lambda pour les écritures dans la région locale. Cela facilite les opérations d'insertion et de mise à jour, mais pas les opérations de suppression.
- Les tables globales configurées pour une cohérence à terme entre plusieurs régions (tables MREC) répliquent les modifications en lisant ces modifications à partir d'un flux DynamoDB sur une table de réplica et en appliquant cette modification à toutes les autres tables de réplica. DynamoDB est donc activé par défaut sur tous les réplicas d'une table globale MREC et ne peut pas être désactivé sur ces réplicas. Le processus de réplication MREC peut combiner plusieurs modifications en un court laps de temps, en une seule opération d'écriture répliquée. Par conséquent, le flux de chaque réplica peut contenir des enregistrements légèrement différents. Les enregistrements DynamoDB Streams sur les réplicas MREC sont toujours classés par article, mais l'ordre entre les éléments peut différer selon les réplicas.

- Les tables globales configurées pour une forte cohérence multi-région (tables MRSC) n'utilisent pas DynamoDB Streams pour la réplication. Cette fonctionnalité n'est donc pas activée par défaut sur les réplicas MRSC. Vous pouvez activer DynamoDB Streams sur un réplica MRSC. Les enregistrements DynamoDB Streams sur les réplicas MRSC sont identiques pour tous les réplicas et sont toujours classés par article, mais l'ordre entre les éléments peut différer selon les réplicas.

Comment les tables globales gèrent-elles les transactions ?

- Les opérations transactionnelles sur les tables MRSC généreront des erreurs.
- Les opérations transactionnelles sur les tables MREC offrent des garanties d'atomicité, de cohérence, d'isolation et de durabilité (ACID) uniquement dans la région de laquelle provient l'opération d'écriture. Les transactions ne sont pas prises en charge entre les régions dans les tables globales. Par exemple, si vous avez une table globale MREC avec des réplicas dans les régions USA Est (Ohio) et USA Ouest (Oregon), et que vous réalisez une opération `TransactWriteItems` dans la région USA Est (Ohio), vous remarquerez peut-être des transactions partiellement incomplètes dans la région USA Ouest (Oregon) lorsque les changements sont répliqués. Les modifications seront uniquement répliquées sur les autres régions une fois validées dans la région source.

Comment les tables globales interagissent-elles avec le cache de DynamoDB Accelerator (DAX) ?

Les tables globales contournent le DAX en mettant directement à jour DynamoDB. Ainsi, DAX ne sait pas qu'il contient des données obsolètes. Le cache DAX n'est actualisé que lorsque la durée de vie du cache expire.

Les balises présentes sur les tables se propagent-elles ?

Non, les balises ne se propagent pas automatiquement.

Dois-je sauvegarder des tables dans toutes les régions ou dans une seule ?

La réponse dépend de l'objectif de la sauvegarde.

- Si vous souhaitez garantir la durabilité des données, DynamoDB fournit déjà cette garantie. Le service garantit la durabilité.
- Si vous souhaitez conserver un instantané des enregistrements historiques (par exemple, pour répondre à des exigences réglementaires), une sauvegarde dans une région doit suffire. Vous pouvez copier la sauvegarde vers d'autres régions en utilisant AWS Backup.

- Si vous souhaitez récupérer des données supprimées ou modifiées par erreur, utilisez DynamoDB point-in-time recovery (PITR) dans une région.

Comment déployer des tables globales à l'aide de CloudFormation ?

- CloudFormation représente une table DynamoDB et une table globale sous la forme de deux ressources distinctes : et. `AWS::DynamoDB::Table` `AWS::DynamoDB::GlobalTable` Une méthode consiste à créer toutes les tables susceptibles d'être globales à l'aide de la construction `GlobalTable`, à les conserver dans un premier temps sous forme de tables autonomes et à ajouter des régions ultérieurement, si nécessaire.
- Dans CloudFormation, chaque table globale est contrôlée par une seule pile, dans une seule région, quel que soit le nombre de répliques. Lorsque vous déployez votre modèle, il CloudFormation crée et met à jour toutes les répliques dans le cadre d'une opération de pile unique. Vous ne devez pas déployer la même ressource AWS::DynamoDB::GlobalTable dans plusieurs régions. Cette opération n'est pas prise en charge et entraînera des erreurs. Si vous déployez votre modèle d'application dans plusieurs régions, vous pouvez utiliser des conditions pour ne créer la ressource `AWS::DynamoDB::GlobalTable` que dans une seule région. Vous pouvez également choisir de définir vos ressources `AWS::DynamoDB::GlobalTable` dans une pile distincte de votre pile d'applications et vous assurer qu'elle n'est déployée que dans une seule région.
- Si vous avez une table normale et que vous souhaitez la convertir en table globale tout en la gérant, définissez la politique de suppression sur`Retain`, supprimez la table de la pile, convertissez-la en table globale dans la console, puis importez la table globale en tant que nouvelle ressource dans la pile. CloudFormation Pour plus d'informations, consultez le AWS GitHub référentiel.
- La réplication entre comptes n'est pas prise en charge pour le moment.

Conclusion et ressources

Les tables globales DynamoDB comportent très peu de contrôles, mais elles nécessitent tout de même un examen attentif. Vous devez déterminer votre mode d'écriture, votre modèle de routage et vos processus d'évacuation. Vous devez équiper votre application dans chaque région et être prêt à ajuster votre routage ou à effectuer une évacuation pour préserver l'état global. L'avantage est que vous disposez d'un jeu de données distribué dans le monde entier avec des opérations de lecture et d'écriture à faible latence, conçu pour une disponibilité de 99,999 %.

Pour plus d'informations sur les tables globales DynamoDB, consultez les ressources suivantes :

- Documentation DynamoDB
- Amazon Application Recovery Controller
- Vérification de l'état de préparation dans ARC (AWS documentation)
- Politiques de routage 53
- AWS Accélérateur mondial
- Contrat de niveau de service DynamoDB
- AWS Principes fondamentaux de plusieurs régions (AWS livre blanc)
- Modèles de conception de résilience des données avec AWS(présentationAWS re:Invent 2022)
- Comment Fidelity Investments et Reltio se sont modernisés avec Amazon DynamoDB AWS(présentation re:Invent 2022)
- Modèles de conception multirégionaux et meilleures pratiques (présentationAWS re:Invent 2022)
- Architecture de reprise après sinistre (DR) activée AWS, partie III : veilleuse et veille chaude (article de AWS blog)
- Utiliser l'épinglage régional pour définir une région d'origine pour les éléments d'une table globale AWS Amazon DynamoDB (article de blog)
- Surveillance d'Amazon DynamoDB à des fins de connaissance AWS opérationnelle (article de blog)
- Mise à l'échelle de DynamoDB : comment les partitions, les raccourcis clavier et le fractionnement ont un impact sur les performances AWS(article de blog)
- Forte cohérence multirégionale avec les tables AWS globales de DynamoDB (présentation re:Invent 2024)

Bonnes pratiques pour gérer le plan de contrôle dans DynamoDB

> **Note**
>
> DynamoDB introduit une limite du plan de contrôle à 2 500 demandes par seconde avec la possibilité de réessayer. Vous trouverez pour plus de détails ci-dessous.

Les opérations du plan de contrôle de DynamoDB vous permettent de gérer les tables DynamoDB ainsi que les objets qui dépendent de tables, tels que les index. Pour plus d'informations sur ces opérations, consultez Plan de contrôle.

Dans certaines circonstances, vous devrez peut-être prendre des mesures et utiliser les données renvoyées par les appels du plan de contrôle dans le cadre de votre logique métier. Par exemple, vous pouvez avoir besoin de connaître la valeur de `ProvisionedThroughput` renvoyée par `DescribeTable`. Dans ces circonstances, suivez les bonnes pratiques suivantes :

- N'interrogez pas trop le plan de contrôle DynamoDB.
- Ne mélangez pas les appels du plan de contrôle et les appels du plan de données dans le même code.
- Gérez les limitations sur les demandes du plan de contrôle et réessayez avec une interruption.
- Invoquez et suivez les modifications apportées à une ressource particulière à partir d'un seul client.
- Au lieu de récupérer les données de la même table plusieurs fois à de courts intervalles, mettez les données en cache pour les traiter.

Bonnes pratiques liées à l'utilisation d'opérations de données groupées dans DynamoDB

DynamoDB prend en charge les opérations groupées, par exemple `BatchWriteItem`, grâce auxquelles vous pouvez effectuer jusqu'à 25 demandes `PutItem` et `DeleteItem` ensemble. Cependant, `BatchWriteItem` ne prend pas en charge les opérations `UpdateItem`. Lorsqu'il s'agit de mises à jour groupées, la distinction réside dans les exigences et la nature de la mise à jour. Vous pouvez utiliser d'autres DynamoDB APIs , par exemple pour une taille de lot `TransactWriteItems` allant jusqu'à 100. Lorsque d'autres éléments sont concernés, vous pouvez utiliser des services tels qu'Amazon EMR AWS Glue,AWS Step Functions ou utiliser des scripts et des outils personnalisés tels que DynamoDB-Shell pour les mises à jour groupées.

Rubriques

- Mise à jour de lot conditionnelle
- Opérations groupées efficaces

Mise à jour de lot conditionnelle

DynamoDB prend en charge les opérations groupées, par exemple `BatchWriteItem`, grâce auxquelles vous pouvez effectuer jusqu'à 25 demandes `PutItem` et `DeleteItem` en un seul lot. Cependant, `BatchWriteItem` ne prend pas en charge les opérations `UpdateItem` ni les expressions de condition. Pour contourner le problème, vous pouvez utiliser d'autres APIs DynamoDB, par exemple pour une taille de lot allant `TransactWriteItems` jusqu'à 100.

Lorsque d'autres éléments sont concernés et qu'une grande partie des données doit être modifiée, vous pouvez utiliser des services tels qu'Amazon EMR AWS GlueAWS Step Functions, ou utiliser des scripts et des outils personnalisés tels que DynamoDB-Shell pour des mises à jour groupées efficaces.

Quand utiliser ce modèle

- DynamoDB-Shell n'est pas un cas d'utilisation pris en charge pour la production.
- `TransactWriteItems` : jusqu'à 100 mises à jour individuelles avec ou sans conditions, exécutées sous la forme d'un bundle ACID « tout ou rien ». Les appels `TransactWriteItems` peuvent également être fournis avec un `ClientRequestToken` si votre application nécessite une idempotence, ce qui signifie que plusieurs appels identiques ont le même effet qu'un seul appel. Cela garantit que vous n'exécuterez pas la même transaction plusieurs fois et que vous n'obtiendrez pas un état de données incorrect.

  Compromis — Un débit supplémentaire est consommé. 2 WCUs par 1 Ko d'écriture au lieu de 1 WGU standard par 1 Ko d'écriture.
- PartiQL `BatchExecuteStatement` : jusqu'à 25 mises à jour avec ou sans conditions. `BatchExecuteStatement` renvoie toujours une réponse positive à la demande globale et renvoie également une liste de réponses aux opérations individuelles qui préserve l'ordre.

  Compromis : pour les lots plus importants, une logique supplémentaire côté client est requise pour distribuer les demandes par lots de 25. Les réponses d'erreur individuelles doivent être prises en compte pour déterminer la stratégie de nouvelle tentative.

Exemples de code

Ces exemples de code utilisent la bibliothèque boto3, qui est le AWS SDK pour Python. Cet exemple suppose que boto3 est installé et configuré avec les informations d'identification AWS appropriées.

Supposons une base de données d'inventaire pour un fournisseur d'appareils électriques qui possède plusieurs entrepôts dans des villes européennes. À l'approche de la fin de l'été, le vendeur aimerait se débarrasser des ventilateurs de bureau pour faire de la place pour d'autres stocks. Le vendeur souhaite offrir une réduction sur tous les ventilateurs de bureau fournis par des entrepôts en Italie, mais uniquement s'il dispose d'un stock de réserve de 20 ventilateurs de bureau. La table DynamoDB est appelée inventaire. Elle possède un schéma clé de partition sku, qui est un identifiant unique pour chaque produit, et un entrepôt de clé de tri, qui est un identifiant d'entrepôt.

Le code Python suivant montre comment effectuer cette mise à jour conditionnelle par lots à l'aide d'un appel d'API `BatchExecuteStatement`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.batch_execute_statement(
        Statements=[
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITTUR1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM5'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
```

```
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN3'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

L'exécution produit le résultat ci-dessous sur des exemples de données :

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '40'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '35'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '33'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '38'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '30'}, 'sku': {'S': 'F123'}}]
```

Comme il s'agit d'une opération limitée pour un système interne, les exigences d'idempotence n'ont pas été prises en compte. Il est possible de placer des barrières de protection supplémentaires, comme la mise à jour des prix uniquement si le prix est supérieur à 35 et inférieur à 40 pour rendre les mises à jour plus robustes.

Nous pouvons également effectuer la même opération de mise à jour groupée en utilisant `TransactWriteItems` en cas d'idempotence et d'exigences ACID plus strictes. Cependant, il est important de se rappeler que soit toutes les opérations du lot de transactions sont effectuées, soit l'ensemble du lot échoue.

Supposons qu'il y ait une vague de chaleur en Italie et que la demande de ventilateurs de bureau ait fortement augmenté. Le fournisseur souhaite augmenter de 20 euros le coût de ses ventilateurs de bureau sortant de tous ses entrepôts en Italie, mais l'organisme de réglementation n'autorise cette augmentation des coûts que si le coût actuel est inférieur à 70 euros pour l'ensemble de son inventaire. Il est essentiel que le prix soit mis à jour dans l'ensemble de l'inventaire en une seule fois et uniquement si le coût est inférieur à 70 euros dans chacun de ses entrepôts.

Le code Python suivant montre comment effectuer cette mise à jour groupée à l'aide d'un appel d'API `TransactWriteItems`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.transact_write_items(
        ClientRequestToken='UUIDAWS124',
        TransactItems=[
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITTUR1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
```

```
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM5'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN3'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

L'exécution produit le résultat ci-dessous sur des exemples de données :

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '60'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '55'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '53'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '55'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '50'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '80'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '75'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '73'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '75'}, 'sku': {'S': 'F123'}},
```

```
{'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '78'}, 'sku':
{'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
{'N': '78'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
'WITVEN3'}, 'price': {'N': '70'}, 'sku': {'S': 'F123'}}]
```

Plusieurs approches permettent d'effectuer des mises à jour groupées dans DynamoDB. L'approche appropriée dépend de facteurs tels que les exigences d' and/or idempuissance de l'ACID, le nombre d'éléments à mettre à jour et la familiarité avec ceux-ci. APIs

Opérations groupées efficaces

Quand utiliser ce modèle

Ces modèles sont utiles pour effectuer efficacement des mises à jour groupées sur des éléments DynamoDB.

- DynamoDB-Shell n'est pas un cas d'utilisation pris en charge pour la production.
- `TransactWriteItems` : jusqu'à 100 mises à jour individuelles avec ou sans conditions, exécutées sous la forme d'un lot ACID « tout ou rien »

  Compromis — Un débit supplémentaire est consommé, 2 WCUs par 1 Ko d'écriture.
- PartiQL `BatchExecuteStatement` : jusqu'à 25 mises à jour avec ou sans conditions

  Compromis : une logique supplémentaire est requise pour distribuer les demandes par lots de 25.
- AWS Step Functions— opérations groupées à débit limité pour les développeurs qui les connaissent bien.AWS Lambda

  Compromis : le temps d'exécution est inversement proportionnel à la limite de débit. Limité par le délai d'expiration maximal de la fonction Lambda. Cette fonctionnalité implique que les modifications de données qui se produisent entre la lecture et l'écriture peuvent être remplacées. Pour plus d'informations, consultez Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- AWS Glue et Amazon EMR : opération en masse à débit limité avec parallélisme géré. Pour les applications ou les mises à jour qui ne sont pas urgentes, ces options ne peuvent s'exécuter en arrière-plan qu'en consommant un faible pourcentage du débit. Les deux services utilisent le emr-dynamodb-connector pour effectuer des opérations DynamoDB. Ces services effectuent une lecture importante suivie d'une écriture importante d'éléments mis à jour avec une option de limitation de débit.

Compromis : le temps d'exécution est inversement proportionnel à la limite de débit. Avec cette fonctionnalité, les modifications de données qui se produisent entre la lecture et l'écriture peuvent être remplacées. Vous ne pouvez pas lire à partir des index secondaires globaux (GSIs). Consultez Remplissage d'un attribut de temps de vie Amazon DynamoDB à l'aide d'Amazon EMR : partie 2.

- DynamoDB Shell : opérations groupées à débit limité utilisant des requêtes de type SQL. Vous pouvez lire ce qui suit GSIs pour une meilleure efficacité.

  Compromis : le temps d'exécution est inversement proportionnel à la limite de débit. Consultez Opérations groupées limitées en débit dans DynamoDB Shell.

Utilisation du motif

Les mises à jour groupées peuvent avoir des implications financières importantes, en particulier si vous utilisez le mode débit à la demande. Il existe un compromis entre vitesse et coût si vous utilisez le mode de débit provisionné. Un réglage très strict du paramètre de limite de débit peut entraîner un temps de traitement très long. Vous pouvez déterminer approximativement la vitesse de mise à jour à l'aide de la taille moyenne des articles et de la limite de débit.

Vous pouvez également déterminer le débit nécessaire pour le processus en fonction de la durée prévue du processus de mise à jour et de la taille moyenne des éléments. Les références de blog partagées avec chaque modèle fournissent des détails sur la stratégie, la mise en œuvre et les limites de l'utilisation du modèle. Pour plus d'informations, consultez Cost-effective bulk processing with Amazon DynamoDB.

Il existe plusieurs approches pour effectuer des mises à jour groupées sur une table DynamoDB active. L'approche appropriée dépend de facteurs tels que les exigences en matière d' and/or idempuissance de l'ACID, le nombre d'éléments à mettre à jour et le niveau de familiarité avec celui-ci. APIs Il est important de prendre en compte le compromis entre le coût et le temps. La plupart des approches décrites ci-dessus offrent une option permettant de limiter le débit utilisé par la tâche de mise à jour groupée.

Bonnes pratiques relatives à la mise en œuvre du contrôle de version dans DynamoDB

Dans les systèmes distribués tels que DynamoDB, le contrôle de version des éléments à l'aide d'un verrouillage optimiste permet d'éviter les mises à jour contradictoires. En suivant les versions des

éléments et en utilisant des écritures conditionnelles, les applications peuvent gérer les modifications simultanées, garantissant ainsi l'intégrité des données dans les environnements à haute simultanéité.

Le verrouillage optimiste est une stratégie utilisée pour garantir que les modifications des données sont appliquées correctement sans conflits. Au lieu de verrouiller les données lorsqu'elles sont lues (comme dans le cas d'un verrouillage pessimiste), le verrouillage optimiste vérifie si les données ont changé avant de les réécrire. Dans DynamoDB, cela est possible grâce à une forme de contrôle de version, dans laquelle chaque élément inclut un identifiant qui augmente à chaque mise à jour. Lors de la mise à jour d'un élément, l'opération n'aboutira que si cet identifiant correspond à celui attendu par votre application.

Quand utiliser ce modèle

Ce modèle est utile dans les cas suivants :

- Plusieurs utilisateurs ou processus peuvent tenter de mettre à jour le même élément simultanément.
- Il est essentiel de garantir l'intégrité et la cohérence des données.
- Il est aussi nécessaire d'éviter les frais généraux et la complexité liés à la gestion des verrous distribués.

En voici quelques exemples :

- Applications de commerce électronique où les niveaux d'inventaire sont fréquemment mis à jour.
- Plateformes collaboratives où plusieurs utilisateurs modifient les mêmes données.
- Systèmes financiers dans lesquels les enregistrements des transactions doivent rester cohérents.

Compromis

Bien que le verrouillage optimiste et les contrôles conditionnels garantissent une intégrité robuste des données, ils comportent les inconvénients suivants :

Conflits de simultanéité

Dans les environnements à simultanéité élevée, le risque de conflits augmente, ce qui peut entraîner de nouvelles tentatives et des coûts d'écriture plus élevés.

Complexité d'implémentation

L'ajout d'un contrôle de version aux éléments et la gestion des vérifications conditionnelles peuvent complexifier la logique de l'application.

Frais de stockage supplémentaires

Le stockage des numéros de version de chaque article augmente légèrement les exigences de stockage.

Conception de modèle

Pour mettre en œuvre ce modèle, le schéma DynamoDB doit inclure un attribut de version pour chaque élément. Voici un schéma simple :

- Clé de partition : un identifiant unique pour chaque élément (par ex., `ItemId`).
- Attributs :
  - `ItemId` : identifiant unique de l'élément.
  - `Version` : nombre entier qui représente le numéro de version de l'élément.
  - `QuantityLeft` : stock restant de l'article.

Lorsqu'un élément est créé pour la première fois, l'attribut `Version` est défini sur 1. À chaque mise à jour, le numéro de version augmente de 1.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	1	10
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

Utilisation du motif

Pour implémenter ce modèle, procédez comme suit dans le flux de votre application :

1. Lisez la version actuelle de l'élément.

   Extrayez l'élément actuel de DynamoDB et lisez son numéro de version.

   ```
   def get_document(item_id):
       response = table.get_item(Key={'ItemID': item_id})
       return response['Item']

   document = get_document('Bananas')
   current_version = document['Version']
   ```

2. Incrémentez le numéro de version dans la logique de votre application. Il s'agira de la version attendue pour la mise à jour.

   ```
   new_version = current_version + 1
   ```

3. Essayez de mettre à jour l'élément à l'aide d'une expression conditionnelle pour vous assurer que le numéro de version correspond.

   ```
   def update_document(item_id, qty_bought, current_version):
       try:
           response = table.update_item(
               Key={'ItemID': item_id},
               UpdateExpression="set #qty = :qty, Version = :v",
               ConditionExpression="Version = :expected_v",
               ExpressionAttributeNames={
                   '#qty': 'QuantityLeft'
               },
               ExpressionAttributeValues={
                   ':qty': qty_bought,
                   ':v': current_version + 1,
                   ':expected_v': current_version
               },
               ReturnValues="UPDATED_NEW"
           )
           return response
       except ClientError as e:
           if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
               print("Update failed due to version conflict.")
           else:
               print("Unexpected error: %s" % e)
           return None
   ```

```
update_document('Bananas', 2, new_version)
```

Si la mise à jour est réussie, QuantityLeft le prix de l'article sera réduit de 2.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	2	8
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

4. Gérez les conflits s'ils surviennent.

   En cas de conflit (par exemple, un autre processus a mis à jour l'élément depuis votre dernière lecture), traitez-le de manière appropriée, par exemple en réessayant l'opération ou en alertant l'utilisateur.

   Cela nécessitera une lecture supplémentaire de l'élément à chaque nouvelle tentative. Limitez donc le nombre total de tentatives que vous autorisez avant l'échec complet de la boucle de demande.

```
def update_document_with_retry(item_id, new_data, retries=3):
    for attempt in range(retries):
        document = get_document(item_id)
        current_version = document['Version']

        result = update_document(item_id, qty_bought, current_version)

        if result is not None:
            print("Update succeeded.")
            return result
        else:
            print(f"Retrying update... ({attempt + 1}/{retries})")

    print("Update failed after maximum retries.")
    return None
```

```
update_document_with_retry('Bananas', 2)
```

La mise en œuvre du contrôle de version des éléments à l'aide de DynamoDB avec un verrouillage optimiste et des vérifications conditionnelles constitue un modèle puissant pour garantir l'intégrité des données dans les applications distribuées. Bien qu'elle introduise une certaine complexité et des compromis potentiels en matière de performances, elle est inestimable dans les scénarios nécessitant un contrôle rigoureux de la simultanéité. En concevant soigneusement le schéma et en mettant en œuvre les contrôles nécessaires dans la logique de votre application, vous pouvez gérer efficacement les mises à jour simultanées et maintenir la cohérence des données.

Des conseils et des stratégies supplémentaires concernant la mise en œuvre du contrôle de version de vos données DynamoDB sont disponibles sur le Blog de la base de données AWS.

Bonnes pratiques pour comprendre vos rapports AWS de facturation et d'utilisation dans DynamoDB

Ce document explique les codes de facturation `UsageType` pour les frais liés à DynamoDB.

AWS fournit des rapports sur les coûts et l'utilisation (CUR) contenant des données relatives aux services utilisés. Vous pouvez l'utiliser AWS Cost and Usage Report pour publier des rapports de facturation sur Amazon S3 au format CSV. Lors de la configuration des CUR, vous pouvez choisir de ventiler les périodes par heure, jour ou mois, et vous pouvez choisir de ventiler l'utilisation par ID de ressource ou non. Pour plus de détails sur la génération de CUR, consultez Création de rapports sur les coûts et l'utilisation.

Dans l'export CSV, vous trouverez les attributs pertinents répertoriés pour chaque ligne. Voici des exemples d'attributs pouvant être inclus :

- lineitem/ UsageStartDate : date et heure de début de l'élément de ligne en UTC inclus.
- lineitem/ UsageEndDate : date et heure de fin de l'élément de ligne correspondant en UTC, sauf.
- lineitem/ ProductCode : Pour DynamoDB, ce sera « DB » AmazonDynamo
- lineitem/ UsageType : code de description spécifique pour le type d'utilisation, tel qu'il est énuméré dans ce document
- lineitem/Operation : nom qui fournit le contexte de la charge, tel que le nom de l'opération à l'origine de la charge (facultatif).

- lineitem/ ResourceId : identifiant de la ressource à l'origine de l'utilisation. Disponible si le CUR inclut une ventilation par ID de ressource.
- lineitem/ UsageAmount : quantité d'utilisation encourue au cours de la période spécifiée.
- lineitem/ UnblendedCost : Le coût de cette utilisation.
- lineitem/ LineItemDescription : Description textuelle de l'élément de ligne.

Pour plus d'informations sur le dictionnaire de données CUR, consultez Cost and Usage Report (CUR) 2.0. Notez que les noms exacts varient en fonction du contexte.

Un `UsageType` est une chaîne dont la valeur est, par exemple, `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` ou `USE1-TimedPITRStorage-ByteHrs`. Chaque type d'utilisation commence par un préfixe de région facultatif. En cas d'absence, cela indique la région us-east-1. Le cas échéant, le tableau ci-dessous fait correspondre le code de région de facturation abrégé au code et au nom de région conventionnels.

Par exemple, l'utilisation nommée `USW2-ReadRequestUnits` indique les unités de demande de lecture consommées dans us-west-2.

Code de région de facturation	Code région	Nom de la région
AFS1	af-south-1	Afrique (Le Cap)
APE1	ap-east-1	Asie-Pacifique (Hong Kong)
APN1	ap-northeast-1	Asie-Pacifique (Tokyo)
APN2	ap-northeast-2	Asie-Pacifique (Séoul)
APN3	ap-northeast-3	Asie-Pacifique (Osaka)
APS1	ap-southeast-1	Asie-Pacifique (Singapour)
APS2	ap-southeast-2	Asie-Pacifique (Sydney)
APS3	ap-south-1	Asie-Pacifique (Mumbai)
APS4	ap-southeast-3	Asie-Pacifique (Jakarta)
APS5	ap-south-2	Asie-Pacifique (Hyderabad)

Code de région de facturation	Code région	Nom de la région
APS6	ap-southeast-4	Asie-Pacifique (Melbourne)
CAN1	ca-central-1	Canada (Centre)
UE	eu-west-1	Europe (Irlande)
EUC1	eu-central-1	Europe (Francfort)
EUC2	eu-central-2	Europe (Zurich)
EUN1	eu-north-1	Europe (Stockholm)
EUS1	eu-south-1	Europe (Milan)
EUS2	eu-south-2	Europe (Espagne)
EUW1	eu-west-1	Europe (Irlande)
EUW2	eu-west-2	Europe (Londres)
EUW3	eu-west-3	Europe (Paris)
ILC1	Il-central-1	Israël (Tel Aviv)
MEC1	me-central-1	Moyen-Orient (EAU)
MES1	me-south-1	Middle East (Bahrain)
SAE1	sa-east-1	Amérique du Sud (São Paulo)
USE1 (par défaut)	us-east-1	USA Est (Virginie du Nord)
USE2	us-east-2	USA Est (Ohio)
UGE1	us-gov-east-1	Gouvernement des États-Unis Est
UGW1	us-gov-west-1	Gouvernement des États-Unis Ouest

Code de région de facturation	Code région	Nom de la région
USW1	us-west-1	USA Ouest (Californie du Nord)
USW2	us-west-2	USA Ouest (Oregon)

Dans les sections suivantes, nous utilisons un modèle `REG-UsageType` pour examiner les frais de DynamoDB, où REG indique la région où l'utilisation a eu lieu et usageType est le code du type de charge. Par exemple, si vous voyez un élément correspondant à `USW1- ReadCapacityUnit-Hrs` dans votre fichier CSV, cela signifie que l'utilisation a eu lieu dans US-West-1 pour la capacité de lecture provisionnée. Dans ce cas, la liste indiquerait `REG-ReadCapacityUnit-Hrs`.

Rubriques

- Capacité de débit
- Flux
- Stockage
- Sauvegarde et restauration
- Transfert de données
- CloudWatch Informations sur les contributeurs
- DynamoDB Accelerator (DAX)

Capacité de débit

Capacité provisionnée en lecture et en écriture

Lorsque vous créez une table DynamoDB en mode capacité provisionnée, vous spécifiez la capacité de lecture et d'écriture dont vous pensez que votre application aura besoin. Le type d'utilisation dépend de la classe de votre table (Standard ou Standard - Accès peu fréquent). Vous configurez les lectures et les écritures en fonction du taux de consommation par seconde, mais les frais sont facturés par heure en fonction de la capacité provisionnée.

UsageType	Uunités	Granularité	Description
REG- ReadCapacityUnit -Heures	RCU-hours	Heure	Frais pour les lectures en mode capacité provisionnée à l'aide de la classe de table Standard.
REG-IA-ReadCapacityUnit-Hrs	RCU-hours	Heure	Frais pour les lectures en mode capacité provisionnée à l'aide de la classe de table Standard - IA.
REG- WriteCapacityUnit -Heures	WCU-hours	Heure	Frais pour les écritures en mode capacité provisionnée à l'aide de la classe de table Standard.
REG-IA-WriteCapacityUnit-Hrs	WCU-hours	Heure	Frais pour les écritures en mode capacité provisionnée à l'aide de la classe de table Standard - IA.

Capacité réservée de lecture et d'écriture

Avec une capacité réservée, vous payez une fois un droit initial et vous vous engagez à un niveau d'utilisation alloué minimal sur une période donnée. La capacité réservée est facturée à un taux horaire réduit. Toute capacité que vous allouez au-delà de votre capacité réservée est facturée selon les frais de capacité allouée standard. La capacité réservée est disponible pour les unités de capacité de lecture et d'écriture (RCU et WCU) provisionnées pour une seule région sur les tables DynamoDB qui utilisent la classe de table standard. Les capacités réservées d'un an et de 3 ans sont facturées de la même manière. SKUs

UsageType	Uunités	Granularité	Description
REG-:dynamodb.read HeavyUsage	RCU-hours	À l'avance, puis sur une base mensuelle	Frais pour les lectures à capacité réservée : un montant initial unique et un prélèvement mensuel au début de chaque mois couvrant toutes les heures RCU engagées à prix réduit au cours du mois. Il y aura des articles correspondants à coût REG-ReadC apacityUnit-Hrs nul.
REG-:dynamodb.write HeavyUsage	WCU-hours	À l'avance, puis sur une base mensuelle	Frais pour les écritures à capacité réservée : un montant initial unique et un prélèvement mensuel au début de chaque mois couvrant toutes les heures WCU engagées à prix réduit au cours du mois. Il y aura des articles correspondants à coût REG-WriteCapacityU nit-Hrs nul.

Capacité de lecture et d'écriture à la demande

Lorsque vous créez une table DynamoDB en mode de capacité à la demande, vous payez uniquement les lectures et écritures effectuées par votre application. Les prix des demandes de lecture et d'écriture dépendent de votre classe de table.

UsageType	Uunités	Granularité	Description
REG- ReadRequestUnits	RRUs	Unit	Frais pour les lectures en mode capacité à la demande avec la classe de table Standard.
REG-IA- ReadRequestUnits	RRUs	Unit	Frais pour les lectures en mode capacité à la demande avec la classe de table Standard - IA.
REG- WriteRequestUnits	WRUs	Unit	Frais pour les écritures en mode capacité à la demande avec la classe de table Standard.
REG-IA- WriteRequestUnits	WRUs	Unit	Frais pour les écritures en mode capacité à la demande avec la classe de table Standard - IA.

Lectures et écritures de tables globales

DynamoDB facture l'utilisation des tables globales en fonction des ressources utilisées sur chaque table de réplica. Pour les tables globales provisionnées, les demandes d'écriture pour les tables globales sont mesurées en répliquées WCUs (RWCU) plutôt qu'en standard WCUs et les écritures

sur les index secondaires globaux dans les tables globales sont mesurées en. WCUs Pour les tables globales à la demande, les demandes d'écriture sont mesurées en répliquées WRUs (RWru) plutôt qu'en standard. WRUs Le nombre de r WCUs ou de r WRUs consommés pour la réplication dépend de la version des tables globales que vous utilisez. La tarification dépend de votre classe de table.

Les écritures sur les index secondaires globaux (GSIs) sont facturées en utilisant les unités d'écriture standard (WCUs et WRUs). Les demandes de lecture et le stockage de données sont facturés de la même manière aux tables à région unique.

Si vous ajoutez un réplica de table pour créer ou étendre une table globale dans de nouvelles régions, DynamoDB facture la restauration d'une table dans les régions ajoutées par gigaoctet de données restaurées. Les données restaurées sont facturées comme REG-RestoreDataSize-Bytes. Consultez Sauvegarde et restauration pour DynamoDB pour plus de détails. La réplication entre régions et l'ajout de réplicas aux tables contenant des données entraînent également des frais de transferts de données sortants.

Lorsque vous sélectionnez le mode de capacité à la demande pour vos tables globales DynamoDB, vous ne payez que les ressources utilisées par votre application sur chaque table de réplica.

UsageType	Uunités	Granularité	Description
REG- ReplWrite CapacityUnit -Heures	rWCU-hours	Heure	Table globale, provisionnée, classe de table Standard.
REG-IA- -Heures ReplWriteCapacityU nit	rWCU-hours	Heure	Table globale, provisionnée, classe de table Standard - IA.
REG- ReplWrite RequestUnits	rWRU	Unit	Table globale, à la demande, classe de table Standard.
REG-IA- ReplWrite RequestUnits	rWRU	Unit	Table globale, à la demande, classe de table Standard - IA.

Flux

DynamoDB utilise deux technologies de streaming, DynamoDB Streams et Kinesis. Chacune possède une tarification distincte.

DynamoDB Streams facture la lecture des données dans les unités de demande de lecture. Chaque appel d'API `GetRecords` est facturé comme une demande de lecture de flux. Vous n'êtes pas facturé pour les appels `GetRecords` d'API invoqués dans AWS Lambda le cadre des déclencheurs DynamoDB ou par les tables globales DynamoDB dans le cadre de la réplication.

UsageType	Uunités	Granularité	Description
Reg Streams-RequestsCount	Nombre	Unit	Lisez les unités de requête pour DynamoDB Streams.

Amazon Kinesis Data Streams facture en unités de capture de données de modification. DynamoDB facture une unité de capture de données de modification pour chaque écriture (jusqu'à 1 Ko). Pour les éléments d'une taille supérieure à 1 Ko, des unités de capture de données de modification supplémentaires sont requises. Vous ne payez que pour les écritures effectuées par votre application sans avoir à gérer la capacité de débit sur la table.

UsageType	Uunités	Granularité	Description
REG- ChangeDataCaptureUnits - Kinésis	Unités CDC	Unit	Unités de capture de données de modification pour Kinesis Data Streams.

Stockage

DynamoDB mesure la taille de vos données facturables en ajoutant la taille d'octet brute de vos données plus des frais de stockage par élément qui dépendent des fonctionnalités que vous avez activées.

> **Note**
>
> Les valeurs d'utilisation du stockage dans le CUR seront supérieures par rapport aux valeurs de stockage lors de l'utilisation de `DescribeTable`, car `DescribeTable` n'inclut pas les frais de stockage par élément.

Le stockage est calculé par heure, mais le prix est mensuel, sur la base d'une moyenne des frais horaires.

Bien que le stockage `UsageType` utilise `ByteHrs` comme suffixe, l'utilisation du stockage dans le CUR est mesurée en Go et facturée par Go par mois.

UsageType	Uunités	Granularité	Description
REG- TimedStorage - ByteHrs	Go	Mois	Quantité de stockage utilisée par vos tables et index DynamoDB, pour les tables dotées de la classe de table Standard.
REG-IA- - TimedStorage ByteHrs	Go	Mois	Quantité de stockage utilisée par vos tables et index DynamoDB, pour les tables dotées de la classe de table Standard - IA.

Sauvegarde et restauration

DynamoDB propose deux types de sauvegardes : les sauvegardes à reprise ponctuelle (PITR) et les sauvegardes à la demande. Les utilisateurs peuvent également effectuer des restaurations à partir de ces sauvegardes dans des tables DynamoDB. Les frais ci-dessous concernent à la fois les sauvegardes et les restaurations.

Les frais de stockage des sauvegardes sont facturés le premier du mois et des ajustements sont effectués tout au long du mois à mesure que des sauvegardes sont ajoutées ou supprimées. Consultez le blog Compréhension des sauvegardes et de la facturation à la demande Amazon DynamoDB pour plus d'informations.

UsageType	Uunités	Granularité	Description
REG- TimedBackupStorage - ByteHrs	Go	Mois	Stockage utilisé par les sauvegardes à la demande de vos tables DynamoDB et de vos index secondaires locaux.
Chronométré - PITRStorage ByteHrs	Go	Mois	Stockage utilisé par les sauvegardes point-in-time de restauration (PITR). DynamoDB surveille la taille de vos tables compatibles PITR en continu, tout au long du mois, afin de déterminer vos frais de sauvegarde et vos factures de stockage tant que la PITR est activé.
REG- RestoreDataSize -Octets	Go	Size	Taille totale des données restaurées (y compris les données de table, les index secondaires locaux et les index secondaires globaux) mesurée en Go à

UsageType	Uunités	Granularité	Description
			partir de sauvegardes DynamoDB.

AWS Backup

AWS Backup est un service de sauvegarde entièrement géré qui facilite la centralisation et l'automatisation de la sauvegarde des données entre les AWS services dans le cloud et sur site.AWS Backup est facturé pour le stockage (stockage à chaud ou à froid), les activités de restauration et le transfert de données entre régions. Les `UsageType` frais suivants apparaissent sous la rubrique « AWS Backup » ProductCode plutôt que sous la rubrique « AmazonDynamo DB ».

UsageType	Uunités	Granularité	Description
REG- WarmStorage - ByteHrs -DynamoDB	Go	Mois	Stockage utilisé par les sauvegardes DynamoDB gérées par AWS Backup tout au long du mois, mesuré en Go par mois.
REG- CrossRegion - WarmBytes - DynamoDB	Go	Size	Les données sont transférées vers une autre AWS région, soit au sein du même compte, soit vers un autre AWS compte. Des frais de transferts entre régions sont facturés lors de la copie de sauvegardes d’une région vers une autre région. Les frais sont toujours facturés sur le compte à partir

UsageType	Uunités	Granularité	Description
			duquel les données sont transférées.
Reg-Restore- -DynamoDB WarmBytes	Go	Size	Taille totale des données restaurées à partir du stockage chaud, mesuré en Go.
REG- ColdStorage - ByteHrs -DynamoDB	Go	Mois	Stockage à froid utilisé par les sauvegardes DynamoDB gérées par AWS Backup tout au long du mois, mesuré en Go par mois.
Reg-Restore- - DynamoDB ColdBytes	Go	Mois	Taille totale des données restaurées à partir du stockage à froid, mesuré en Go.

Exportation et importation

Vous pouvez exporter des données de DynamoDB vers Amazon S3 ou importer des données d'Amazon S3 vers une nouvelle table DynamoDB.

Bien que `UsageType` utilise `Bytes` en tant que suffixe, l'utilisation des exportations et des importations dans le CUR est mesurée et facturée en Go.

UsageType	Uunités	Granularité	Description
REG- ExportDataSize -Octets	Go	Size	Les frais d'exportation de données vers S3. DynamoDB facture les données que vous

UsageType	Uunités	Granularité	Description
			exportez en fonction de la taille de la table DynamoDB (données de table et index secondaires locaux) au moment spécifié lorsque l'exportation a été créée.
REG- ImportDataSize -Octets	Go	Size	Frais d'importation de données à partir de S3. La taille est calculée en fonction de la taille de l'objet non compressé des données dans Amazon S3. Il n'y a pas de frais supplémentaires pour l'importation vers des tables avec GSIs.
REG- Increment alExportDataSize - Octets	Go	Size	Les frais liés à la taille des données traitées à partir de la sauvegard e continue pour produire des exportati ons incrémentielles.

Transfert de données

L'activité de transfert de données peut apparaître associée au service DynamoDB. DynamoDB ne facture pas le transfert de données entrantes, ni le transfert de données entre DynamoDB et d'AWS autres services de la AWS même région (en d'autres termes, 0,00 USD par Go). Les données

transférées entre AWS les régions (par exemple entre DynamoDB dans la région USA Est [Virginie du Nord] et EC2 Amazon dans la région UE [Irlande]) sont facturées des deux côtés du transfert.

UsageType	Uunités	Granularité	Description
REG- DataTransfer - En octets	Go	Uunités	Données transféré es vers DynamoDB depuis Internet.
REG- DataTransfer - Octets de sortie	Go	Uunités	Données transférées depuis DynamoDB vers Internet.

CloudWatch Informations sur les contributeurs

CloudWatch Contributor Insights for DynamoDB est un outil de diagnostic permettant d'identifier les clés les plus fréquemment consultées et les plus limitées de votre table DynamoDB. Les `UsageType` frais suivants apparaissent sous le « AmazonCloudWatch » ProductCode plutôt que sous le « AmazonDynamo DB ».

UsageType	Uunités	Granularité	Description
REG-CW : Contribut orEventsManaged	Événements traités	Uunités	Nombre d’événeme nts DynamoDB traités. Par exemple, pour un tableau dans lequel CloudWatch Contributor Insights est activé, chaque fois qu'un élément est lu ou écrit, il est compté comme un événement . Si la table comporte une clé de tri, cela entraîne des frais

UsageType	Uunités	Granularité	Description
			pour deux événement s.
REG-CW : Contribut orRulesManaged	Nombre de règles	Mois	DynamoDB crée des règles pour identifie r les éléments les plus consultés et les clés les plus limitées lorsque vous activez Cloud Watch Contributor Insights. Ces frais sont facturés pour les règles ajoutées pour chaque entité (tables et GSIs) configuré e pour enregistrer les informations des CloudWatch contribut eurs.

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) est facturé à l'heure en fonction du type d'instance sélectionné pour le service. Les frais ci-dessous se réfèrent aux instances DynamoDB Accelerator provisionnées. Les `UsageType` frais suivants apparaissent sous « AmazonDAX » ProductCode plutôt que sous « AmazonDynamo DB ».

UsageType	Uunités	Granularité	Description
REG-:dax-NodeUsage <INSTANCETYPE>	Node-hour	Heure	L'utilisation horaire d'un type d'instance particulier. La tarificat ion est calculée par heure de nœud

UsageType	Uunités	Granularité	Description
			consommée, à partir du moment où un nœud est lancé jusqu'à sa fermeture . Chaque heure de nœud partielle consommée sera facturée comme une heure complète. DAX facture chaque nœud d'un cluster DAX. Si vous avez un cluster comportant plusieurs nœuds, vous verrez plusieurs éléments dans votre rapport de facturation.

Le type d'instance sera l'une des valeurs de la liste suivante. Pour plus de détails sur les types de nœud, consultez Nœuds.

- r3.2xlarge, r4.8xlarge ou r5.8xlarge
- r3.4xlarge, r4.large ou r5.large
- r3.8xlarge, r4.xlarge ou r5.xlarge
- r3.2xlarge, r5.12xlarge ou t2.medium
- r3.4xlarge, r4.large ou r5.large
- r3.xlarge, r5.16xlarge ou t2.small
- r4.16xlarge, r5.24xlarge ou t3.medium
- r4.2xlarge, r5.2xlarge ou t3.small
- r4.4xlarge ou r5.4xlarge

Migration d’une table DynamoDB d’un compte à un autre

Vous pouvez migrer une table Amazon DynamoDB d’un compte à un autre afin de mettre en œuvre une stratégie multi-comptes ou de sauvegarde. Vous pouvez également le faire pour des raisons de test, de débogage ou de conformité. Un cas d'utilisation courant consiste à copier des tables DynamoDB dans des environnements de production, de préparation, de test et de développement dans lesquels chaque environnement utilise un compte différent.AWS

DynamoDB propose deux options pour migrer des tables d'un compte vers AWS un autre :

- AWS Backup pour la sauvegarde et la restauration entre comptes :AWS Backup est un service de sauvegarde entièrement géré qui vous permet de gérer de manière centralisée les sauvegardes sur plusieurs AWS services. Grâce à ses fonctionnalités de sauvegarde et de restauration entre comptes, vous pouvez sauvegarder une table DynamoDB dans un compte et restaurer la sauvegarde sur un autre compte de la même organisation AWS.
- Exportation et importation DynamoDB vers Amazon S3 : l'utilisation des fonctionnalités d'exportation et d'importation de DynamoDB vers Amazon S3 vous permet d'effectuer une exportation complète vers un compartiment Amazon S3, puis d'importer ces données dans une nouvelle table d'un autre compte.AWS Cette approche convient lorsque vous devez effectuer une migration entre des comptes qui ne font pas partie de la même AWS organisation ou si vous ne souhaitez pas les utiliser AWS Backup.

> **Note**
>
> L'importation depuis Amazon S3 ne prend pas en charge les tables avec des index secondaires locaux (LSIs), mais elle prend en charge les index secondaires globaux ()GSIs. Pour plus d'informations sur LSIs et GSIs, voirAmélioration de l’accès aux données avec les index secondaires dans DynamoDB.

Rubriques

- Migrer une table à des AWS Backup fins de sauvegarde et de restauration entre comptes
- Migration d’une table à l’aide de l’exportation vers S3 et de l’importation depuis S3

Migrer une table à des AWS Backup fins de sauvegarde et de restauration entre comptes

Conditions préalables

- AWS Les comptes source et cible doivent appartenir à la même organisation dans le service AWS Organizations
- Autorisations valides de gestion des AWS identités et des accès (IAM) pour créer et utiliser des coffres-forts AWS Backup

Pour plus d'informations sur la configuration des sauvegardes entre comptes, voir Création de copies de sauvegarde entre AWS comptes.

Informations sur la tarification

AWS les frais de sauvegarde (en fonction de la taille de la table), de toute copie de données entre AWS les régions (en fonction de la quantité de données), de restauration (en fonction de la quantité de données) et de tous les frais de stockage permanents. Pour éviter des frais récurrents, vous pouvez supprimer la sauvegarde si vous n'en avez pas besoin après la restauration.

Pour plus d'informations sur la tarification, consultez Tarification d'AWS Backup.

Étape 1 : Activer les fonctionnalités avancées pour DynamoDB et la sauvegarde entre comptes

1. Dans le compte source et le AWS compte cible, accédez à la console AWS de gestion et ouvrez la console AWS Backup.
2. Choisissez l'option Paramètres.
3. Sous Fonctionnalités avancées pour les sauvegardes Amazon DynamoDB, vérifiez que les fonctionnalités avancées sont activées. Si ce n'est pas le cas, choisissez Activer.
4. Sous Gestion entre comptes, pour la Sauvegarde entre comptes, sélectionnez Activer.

Étape 2 : Créer un coffre de sauvegarde dans le compte source et le compte cible

1. Dans les AWS comptes source, ouvrez la console AWS Backup.
2. Choisissez Coffres-forts de sauvegarde.
3. Choisissez Créer un coffre-fort de sauvegarde.

4. Copiez et enregistrez le nom de ressource Amazon (ARN) des coffres-forts de sauvegarde créés et du AWS compte cible.
5. Vous aurez besoin des coffres-forts ARNs de sauvegarde source et cible pour copier la sauvegarde de la table DynamoDB entre les comptes.

Étape 3 : Créer une sauvegarde de table DynamoDB dans le compte source

1. Sur la page du Tableau de bord de sauvegarde AWS, choisissez Créer une sauvegarde à la demande.
2. Dans la section Paramètres, sélectionnez DynamoDB comme Type de ressource, puis sélectionnez le nom de la table.
3. Dans la liste déroulante Coffre de sauvegarde, sélectionnez le coffre de sauvegarde que vous avez créé dans le compte source.
4. Sélectionnez la Période de rétention souhaitée.
5. Choisissez Create on-demand backup (Créer une sauvegarde à la demande).
6. Surveillez l'état de la tâche de sauvegarde dans l'onglet Tâches de sauvegarde de la page Tâches de sauvegarde AWS.

Étape 4 : Copier la sauvegarde de table DynamoDB du compte source vers le compte cible

1. Une fois le travail de sauvegarde terminé, ouvrez la AWS Backup console dans le compte source et choisissez Backup vaults.
2. Sous Sauvegardes, choisissez la sauvegarde de table DynamoDB. Choisissez Actions, puis Copier.
3. Entrez la AWS région du compte cible.
4. Pour l'ARN du coffre externe, saisissez l'ARN du coffre de sauvegarde que vous avez créé dans le compte cible.
5. Dans le coffre de sauvegarde du compte cible, activez l'accès depuis un compte source pour autoriser la copie des sauvegardes.

Étape 5 : Restaurer la sauvegarde de la table DynamoDB dans le compte cible

1. Dans le AWS compte cible, ouvrez la AWS Backup console et choisissez Backup vaults

2. Sous Sauvegardes, sélectionnez la sauvegarde que vous avez copiée à partir du compte source. Choisissez Actions, puis Restaurer.
3. Saisissez le nom de la nouvelle table DynamoDB, le chiffrement de cette nouvelle table, la clé avec laquelle vous souhaitez chiffrer la restauration et d'autres options.
4. Lorsque la restauration est terminée, le statut de la table indique Actif.

Migration d'une table à l'aide de l'exportation vers S3 et de l'importation depuis S3

Conditions préalables

- Vous devez activer Point-in-Time Recovery (PITR) pour votre table afin d'effectuer l'exportation vers S3. Pour de plus amples informations, veuillez consulter Activation de la reprise ponctuelle pour DynamoDB.
- Validez des autorisations IAM pour effectuer l'exportation. Pour de plus amples informations, veuillez consulter Demande d'exportation de table dans DynamoDB.
- Validez des autorisations IAM suffisantes pour effectuer l'importation. Pour de plus amples informations, veuillez consulter Demande d'importation de table dans DynamoDB.

Informations sur la tarification

AWS frais pour le PITR (en fonction de la taille de la table et de la durée pendant laquelle le PITR est activé). Si vous n'avez pas besoin de la PITR sauf pour l'exportation, vous pouvez la désactiver une fois l'exportation terminée.AWS facture également les demandes adressées à S3, pour le stockage des données exportées dans S3 et pour l'importation (en fonction de la taille non compressée des données importées).

Pour plus d'informations sur la tarification DynamoDB, consultez Tarification de DynamoDB.

> **Note**
>
> La taille et le nombre d'objets sont limités lors de l'importation depuis S3 vers DynamoDB. Pour de plus amples informations, veuillez consulter Quotas d'importation.

Demande d’une exportation de table vers Amazon S3

1. Connectez-vous à la console AWS de gestion et ouvrez la console DynamoDB.
2. Dans le panneau de navigation sur le côté gauche de la console, choisissez Exportations vers S3.
3. Choisissez une table source et un compartiment S3 de destination. Saisissez l’URL du compartiment du compte de destination avec le format `s3://bucketname/prefix`. `/prefix` est un dossier facultatif qui vous permet de garder votre compartiment de destination organisé.
4. Choisissez Exportation complète. Une exportation complète sort l’instantané complet de votre table telle qu’elle était au moment spécifié.
   a. Sélectionnez Heure actuelle pour exporter le dernier instantané complet de la table.
   b. Pour Format de fichier exporté, choisissez entre JSON DynamoDB et Amazon Ion. L’option par défaut est DynamoDB JSON.
5. Cliquez sur le bouton Export (Exporter) pour commencer l’exportation.
6. Les exportations de petites tables devraient s’achever en quelques minutes, mais elles peuvent durer plus d’une heure pour les tables de l’ordre du téraoctet.

Demande d’importation de table à partir d’Amazon S3

1. Connectez-vous à la console AWS de gestion et ouvrez la console DynamoDB.
2. Dans le panneau de navigation sur le côté gauche de la console, choisissez Importer depuis S3.
3. Sur la page qui s’affiche, sélectionnez Import from S3 (Importer depuis S3).
4. Entrez l’URL source d’Amazon S3. Vous pouvez également le trouver en utilisant le bouton Parcourir S3. Le chemin d’accès possède le format `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Spécifiez si vou s êtes lePropriétaire du compartiment S3.
6. Sous Compression de fichier d’importation, sélectionnez GZIP pour qu’il corresponde à l’exportation.
7. Sous Format de fichier d’importation, sélectionnez DynamoDB JSON pour qu’il corresponde à l’exportation.
8. Sélectionnez Suivant. Pour Spécifier les détails de la table, choisissez les options de la nouvelle table qui sera créée pour stocker vos données.

9. Sélectionnez Suivant. Pour Configurer les paramètres de la table, personnalisez les paramètres de table supplémentaires, le cas échéant.
10. Sélectionnez à nouveau Next (Suivant) pour vérifier vos options d'importation, puis cliquez sur Import (Importer) pour commencer la tâche d'importation. Vous verrez votre nouvelle table répertoriée sous Importations depuis S3 avec le statut Importation. Vous ne pouvez pas accéder à votre table pendant cette période. Les petites importations devraient s'achever en quelques minutes, mais elles peuvent durer plus d'une heure pour les tables de l'ordre du téraoctet.
11. Une fois l'importation terminée, le statut indique Actif et vous pouvez commencer à utiliser la table.

Synchronisation des tables pendant la migration

Si vous pouvez suspendre les opérations d'écriture sur la table source pendant la durée de la migration, la source et la sortie doivent correspondre exactement après la migration. Si vous ne pouvez pas suspendre les opérations d'écriture, la table cible sera normalement légèrement en retard par rapport à la source après la migration. Pour récupérer la table source, vous pouvez utiliser le streaming (DynamoDB Streams ou Kinesis Data Streams pour DynamoDB) pour rejouer les écritures effectuées dans la table source depuis la sauvegarde ou l'exportation.

Vous devez commencer à lire les enregistrements du flux avant l'horodatage après avoir exporté la table source vers S3. Par exemple, si l'exportation vers S3 a eu lieu à 14 h 00 et que l'importation vers la table cible s'est terminée à 23 h 00, vous devez lancer la lecture du flux DynamoDB à 13 h 58. Le tableau des options de streaming pour la capture des données de modification récapitule les fonctionnalités de chaque modèle de streaming.

L'utilisation de DynamoDB Streams avec Lambda offre une approche rationalisée pour synchroniser les données entre les tables DynamoDB source et cible. Vous pouvez utiliser une fonction Lambda pour rejouer chaque écriture dans la table cible.

> **Note**
>
> Les éléments sont conservés dans DynamoDB Streams pendant 24 heures. Vous devez donc prévoir de terminer votre sauvegarde et la restauration ou l'exportation et l'importation dans cette fenêtre.

Recommandations pour intégrer DAX aux applications DynamoDB

DynamoDB Accelerator (DAX) est un service de mise en cache compatible avec DynamoDB qui fournit des performances en mémoire rapides pour les applications exigeantes, telles que les applications à lecture intensive. Avec DAX, vous pouvez obtenir des temps de réponse en microsecondes pour accéder aux données fréquemment demandées. Ce guide prescriptif de DynamoDB Accelerator fournit des informations complètes et les bonnes pratiques pour intégrer DAX à vos applications DynamoDB.

Ce guide fournit des connaissances de base à ceux qui découvrent DAX ou qui souhaitent optimiser leurs configurations existantes. Ce guide couvre divers sujets, par exemple, quand utiliser DAX et créer un cluster DAX. Il comprend également des exemples pratiques et des explications détaillées pour vous aider à implémenter efficacement DAX dans vos projets. Enfin, ce guide propose des stratégies avancées que vous devez mettre en œuvre pour optimiser les capacités de mise en cache DAX afin de garantir des applications rapides et pouvant être mises à l'échelle.

Rubriques

- Évaluation de l'adéquation de DAX pour vos cas d'utilisation
- Configuration de votre client DAX
- Configuration de votre cluster DAX
- Dimensionnement de votre cluster DAX
- Déploiement d'un cluster
- Gestion des opérations d'un cluster
- Surveillance de DAX

Évaluation de l'adéquation de DAX pour vos cas d'utilisation

Cette section explique quand et pourquoi utiliser DAX. L'utilisation de ces instructions vous aide à déterminer si l'intégration de DAX à DynamoDB est avantageuse pour les modèles de charge de travail, les exigences de performance et les besoins de cohérence des données. Cette section couvre également les scénarios dans lesquels DAX pourrait ne pas être adapté, par exemple les charges de travail lourdes en écriture et les données rarement consultées.

Dans cette section

- Quand et pourquoi choisir DAX
- Cas dans lesquels il est préférable de ne pas utiliser DAX

Quand et pourquoi choisir DAX

Vous pouvez envisager d'ajouter DAX à votre pile d'applications dans plusieurs scénarios. Par exemple, utilisez DAX pour réduire la latence globale des demandes de lecture par rapport à DynamoDB ou pour minimiser les lectures répétées des mêmes données dans une table. La liste suivante présente des exemples de scénarios dans lesquels vous pouvez tirer parti de l'intégration de DAX à DynamoDB :

- Exigence de haute performance
  - Lectures à faible latence : vous pouvez envisager d'utiliser DAX si votre application a besoin de temps de réponse en microsecondes pour des lectures cohérentes à terme. DAX peut également réduire considérablement le temps de réponse pour accéder aux données lues fréquemment.
- Charges de travail intensives en lecture
  - Applications nécessitant beaucoup de lecture : pour les applications dont le read-to-write ratio est élevé, par exemple 10:1 ou plus, le DAX augmente le nombre d'accès au cache et réduit le volume de données périmées. Cela réduit le nombre de lectures par rapport à une table. Pour éviter de lire des données périmées dans le cache si votre application utilise beaucoup d'écriture, veillez à définir une Utilisation de la durée de vie (TTL) dans DynamoDB inférieure pour le cache.
  - Mise en cache des requêtes courantes : si votre application lit fréquemment les mêmes données, par exemple des produits populaires sur une plateforme de commerce électronique, DAX peut répondre à ces demandes directement à partir de son cache.
- Schémas de trafic en rafale
  - Mise à l'échelle plus fluide des tables : DAX permet d'atténuer les impacts des pics de trafic soudains. DAX fournit une mémoire tampon pour augmenter verticalement la capacité des tables DynamoDB de manière progressive, ce qui réduit le risque de limitation de lecture.
  - Débit de lecture supérieur pour chaque élément : DynamoDB alloue des partitions individuelles pour chaque élément. Cependant, une partition commence à limiter les lectures d'un élément lorsqu'il atteint 3 000 unités de capacité de lecture (RCU). DAX vous permet de mettre à l'échelle les lectures d'un seul élément au-delà de 3 000 RCU.
- Optimisation des coûts
  - Réduction des coûts DynamoDB : la lecture depuis DAX peut réduire le nombre de lectures envoyées à une table DynamoDB, ce qui peut avoir un impact direct sur les coûts. Avec un taux d'accès au cache élevé, le coût réduit de lecture des tables peut dépasser le coût d'un cluster DAX, ce qui se traduit par une nette réduction des coûts.
- Exigences de cohérence des données

- Cohérence éventuelle : DAX prend en charge les lectures cohérentes à terme. DAX convient donc aux cas d'utilisation où une cohérence immédiate n'est pas essentielle.
- Mise en cache par écriture simultanée : les écritures que vous effectuez sur DAX sont des écritures simultanée. Une fois que DAX confirme qu'il a écrit un élément dans DynamoDB, il conserve la version de cet élément dans le cache d'éléments. Ce mécanisme d'écriture simultanée permet de maintenir une plus grande cohérence des données entre le cache et la base de données, mais utilise des ressources supplémentaires du cluster DAX.

Cas dans lesquels il est préférable de ne pas utiliser DAX

Bien que DAX soit puissant, il ne convient pas à tous les scénarios. La liste suivante présente des exemples de scénarios dans lesquels il n'est pas conseiller d'intégrer DAX à DynamoDB :

- Charges de travail gourmandes en écriture : le principal avantage de DAX est d'accélérer les lectures, mais les écritures utilisent plus de ressources DAX que les lectures. Si votre application est principalement gourmande en écriture, les avantages de DAX peuvent être limités.
- Lecture de données peu fréquente : si votre application accède aux données de manière peu fréquente ou à un large éventail de données rarement réutilisées (données à froid), il est probable que vous connaissez un faible cache hit ratio. Dans ce cas, la surcharge liée à la maintenance du cache peut ne pas justifier les gains de performances.
- Lectures ou écritures groupées : si votre application effectue plus d'écritures groupées que d'écritures individuelles, vous devez écrire autour de DAX. En outre, pour les lectures groupées, vous devez exécuter des analyses de table complètes directement sur une table DynamoDB.
- Exigences strictes en matière de cohérence ou de transaction : DAX transmet des lectures et des TransactGetItemsappels très cohérents à une table DynamoDB. Vous devez effectuer ces lectures autour du cluster DAX pour éviter d'utiliser les ressources du cluster. Les éléments lus de cette façon ne seront pas mis en cache ; par conséquent, le routage de ces éléments via DAX ne sert à rien.
- Applications simples avec des exigences de performances modestes : pour les applications présentant des exigences de performances modestes et une tolérance à la latence directe de DynamoDB, la complexité et le coût liés à l'ajout de DAX peuvent ne pas être nécessaires. À lui seul, DynamoDB gère un débit élevé et fournit des performances à un chiffre en millisecondes.
- Besoins d'interrogation complexes allant au-delà de l'accès clé-valeur : DAX est optimisé pour les modèles d'accès clé-valeur. Si votre application nécessite des fonctionnalités de requête

complexes associées à un filtrage complexe, telles que les opérations de Requête et d'Analyse, les avantages de la mise en cache DAX peuvent être limités.

Dans ces situations, utilisez Amazon ElastiCache (Redis OSS) comme alternative. ElastiCache (Redis OSS) prend en charge les structures de données avancées, telles que les listes, les ensembles et les hachages. Il propose également des fonctionnalités telles que Pub/Sub, des index géospatiaux et des scripts.

- Exigences de conformité : actuellement, DAX ne propose pas les mêmes accréditations de conformité que DynamoDB. Par exemple, DAX n'a pas encore obtenu l'accréditation SOC.

Configuration de votre client DAX

Le cluster DAX est un cluster basé sur une instance accessible à l'aide de différents DAX. SDKs Chaque kit SDK fournit aux développeurs des options configurables, telles que requestTimeout et les connexions, pour répondre aux exigences spécifiques des applications.

Lors de la configuration de votre client DAX, il est crucial de prendre en compte l'échelle de votre application cliente, en particulier le ratio d'instances clientes par rapport aux instances de serveur DAX (11 au maximum). Les grands flottes d'instances clientes peuvent générer de nombreuses connexions aux instances du serveur DAX, ce qui risque de les surcharger. Ce guide décrit les bonnes pratiques pour la configuration du client DAX.

Bonnes pratiques

1. Instances clientes : mettez en œuvre des instances clientes singleton pour garantir la réutilisation des instances dans toutes les demandes. Pour plus d'informations sur l'implémentation, consultez the section called "Étape 4 : exécuter un exemple d'application".
2. Délais d'expiration des demandes : bien que les applications nécessitent souvent de faibles délais d'attente pour garantir une latence minimale aux systèmes en amont, la définition de délais d'expiration trop bas peut entraîner des problèmes. Les faibles délais d'attente peuvent déclencher des reconnexions fréquentes aux instances de serveur lorsque les serveurs DAX subissent des pics de latence temporaires. En cas d'expiration du délai, le client DAX met fin à la connexion existante au nœud de serveur et en établit une nouvelle. L'établissement de connexions étant gourmand en ressources, de nombreuses connexions consécutives peuvent surcharger les serveurs DAX. Nous vous recommandons la procédure suivante :
   - Maintien des paramètres de délai de demande par défaut.

- Si des délais d'attente plus courts sont nécessaires, mettez en œuvre des threads d'application distincts avec des valeurs de délai d'expiration inférieures et incluez des mécanismes de nouvelle tentative avec un recul exponentiel.

3. Délai d'expiration de connexion : pour la plupart des applications, nous recommandons de conserver les paramètres de délai d'expiration de connexion par défaut.
4. Connexions simultanées : certaines SDKs, telles que JavaV2, permettent d'ajuster les connexions simultanées au serveur DAX. Considérations clés :
   - Les instances de serveur DAX peuvent gérer jusqu'à 40 000 connexions simultanées.
   - Les paramètres par défaut conviennent à la plupart des cas d'utilisation.
   - Les instances client volumineuses associées à un nombre élevé de connexions simultanées peuvent surcharger les serveurs.
   - Des valeurs de connexions simultanées plus faibles réduisent le risque de surcharge du serveur.
   - Exemple de calcul des performances :
     - En supposant une latence de demande de 1 ms, chaque connexion peut théoriquement traiter 1 000 demandes/seconde.
     - Pour un cluster à 3 nœuds, une seule instance client connectée à tous les nœuds peut traiter 3 000 demandes/seconde.
     - Avec 10 connexions, le client peut traiter environ 30 000 demandes/seconde.

       Recommandation : commencez par des paramètres de connexion simultanée plus faibles et validez-les par le biais de tests de performances avec les modèles de charge de travail de production attendus.

Configuration de votre cluster DAX

Le cluster DAX est un cluster géré, mais vous pouvez ajuster ses configurations en fonction des exigences de votre application. En raison de son étroite intégration avec les opérations de l'API DynamoDB, vous devez prendre en compte les aspects suivants lors de l'intégration de votre application à DAX.

Dans cette section

- Tarification de DAX
- Cache d'éléments et cache de requêtes
- Sélection du paramètre TTL pour les caches

- Mise en cache de plusieurs tables avec un cluster DAX
- Réplication des données dans les tables globales DAX et DynamoDB
- Disponibilité de région DAX
- Comportement de mise en cache DAX

Tarification de DAX

Le coût d'un cluster dépend du nombre et de la taille des nœuds qu'il a provisionnés. Chaque nœud est facturé pour chaque heure d'exécution dans le cluster. Pour plus d'informations, consultez Tarification Amazon DynamoDB.

Les accès au cache n'entraînent aucun coût pour DynamoDB, mais ont un impact sur les ressources du cluster DAX. Les erreurs de cache entraînent des coûts de lecture DynamoDB et nécessitent des ressources DAX. Les écritures entraînent des coûts d'écriture DynamoDB et ont un impact sur les ressources du cluster DAX pour l'utilisation du proxy sur l'écriture.

Cache d'éléments et cache de requêtes

DAX maintient un cache d'éléments et un cache de requêtes. Le fait de comprendre les différences entre ces caches peut vous aider à déterminer les caractéristiques de performance et de cohérence qu'ils offrent à votre application.

Caractéristiques du cache	Cache d'élément	Cache de requête
Objectif	Stocke les résultats GetItemet les opérations BatchGetI tem d'API.	Stocke les résultats des opérations d'API Query et Scan. Ces opérations peuvent renvoyer plusieurs éléments en fonction des conditions de requête au lieu de clés d'éléments spécifiques.
Type d'accès	Utilise un accès basé sur les clés. Lorsqu'une application demande des données en utilisant `GetItem` ou	Utilise un accès basé sur un paramètre. DAX met en cache le jeu de résultats de `Query` et les opérations d'API `Scan`. DAX

Caractéristiques du cache	Cache d'élément	Cache de requête
	`BatchGetItem` , DAX vérifie d'abord le cache des éléments à l'aide de la clé primaire des éléments demandés. Si l'élément est mis en cache et n'a pas expiré, DAX le renvoie immédiatement sans accéder à la table DynamoDB.	traite les demandes suivantes avec les mêmes paramètre s qui incluent les mêmes conditions de requête, table et index, à partir du cache. Cela réduit considérablement les temps de réponse et la consommation de débit de lecture DynamoDB.
Invalidation du cache	DAX réplique automatiq uement les éléments mis à jour dans le cache d'élément s des nœuds du cluster DAX dans les scénarios suivants : • Vous rédigez une mise à jour d'un élément via le cache. • Lisez une version mise à jour d'un élément dans le tableau.	Le cache de requêtes est plus difficile à invalider que le cache d'éléments. Les mises à jour des éléments peuvent ne pas mapper directement aux requêtes ou aux analyses mises en cache. Vous devez régler avec soin la TTL du cache de requêtes pour garantir la cohérence des données. Les écritures via DAX ou la table de base ne sont pas reflétées dans le cache de requêtes tant que la TTL ne fait pas expirer la réponse précédemment mise en cache et que DAX n'exécute pas une nouvelle requête sur DynamoDB.

Caractéristiques du cache	Cache d'élément	Cache de requête
GSI	Comme l'opération d'API `GetItem` n'est pas prise en charge sur les index secondaires locaux ou les index secondaires globaux, le cache d'éléments met uniquement en cache les lectures de la table de base.	Le cache de requêtes met en cache les requêtes portant à la fois sur les tables et sur les index.

Sélection du paramètre TTL pour les caches

La TTL détermine la période pendant laquelle les données sont stockées dans le cache avant de devenir obsolètes. Après cette période, les données sont automatiquement actualisées lors de la prochaine demande. Pour choisir le bon paramètre TTL pour vos caches DAX, vous devez trouver un équilibre entre l'optimisation des performances des applications et la cohérence des données. Comme il n'existe pas de paramètre TTL universel qui fonctionne pour toutes les applications, le paramètre TTL optimal varie en fonction des caractéristiques et des exigences spécifiques de votre application. Nous vous conseillons de commencer avec un paramètre TTL prudent, en suivant ces recommandations. Ajustez ensuite de manière itérative votre paramètre TTL en fonction des données de performance et des informations de votre application.

DAX gère une liste LRU (moins récemment utilisé) pour le cache d'éléments. La liste LRU indique le moment où les éléments ont été écrits pour la première fois ou lus pour la dernière fois depuis le cache. Si la mémoire du nœud DAX est pleine, DAX expulse les anciens éléments même s'ils n'ont pas encore expiré, afin de ménager de la place pour les nouveaux. L'algorithme LRU est toujours activé et ne peut pas être configuré par l'utilisateur.

Pour définir une durée TTL adaptée à vos applications, tenez compte des points suivants :

Compréhension de vos modèles d'accès aux données

- Charges de travail intensives en lecture : pour les applications présentant des charges de travail lourdes en lecture et des mises à jour de données peu fréquentes, définissez une durée TTL plus longue afin de réduire le nombre d'erreurs de cache. Une durée TTL plus longue réduit également le besoin d'accéder à la table DynamoDB sous-jacente.

- Charges de travail intensives en écriture : pour les applications soumises à des mises à jour fréquentes qui ne sont pas écrites via DAX, définissez une durée TTL plus courte afin de garantir la cohérence du cache avec la base de données. Une durée TTL plus courte réduit également le risque de diffusion de données périmées.

Évaluation des exigences de performance de votre application

- Sensibilité à la latence : si votre application nécessite une faible latence par rapport à la fraîcheur des données, utilisez une durée TTL plus longue. Une durée TTL plus longue optimise les accès au cache, ce qui réduit la latence de lecture moyenne.
- Débit et capacité de mise à l'échelle : une durée TTL plus longue réduit la charge sur les tables DynamoDB et améliore le débit et la capacité de mise à l'échelle. Cependant, vous devez trouver un équilibre entre cela et le besoin de up-to-date données.

Analyse de l'éviction du cache et de l'utilisation de la mémoire

- Limites de mémoire cache : surveillez l'utilisation de la mémoire de votre cluster DAX. Une durée TTL plus longue peut stocker davantage de données dans le cache, ce qui peut atteindre les limites de mémoire et entraîner des évictions basées sur des LRU.

Utilisation des métriques et de la surveillance pour ajuster la TTL

Passez régulièrement en revue les métriques, par exemple les accès et les échecs du cache, ainsi que l'utilisation du processeur et de la mémoire. Ajustez votre paramètre TTL en fonction de ces métriques afin de trouver un équilibre optimal entre les performances et l'actualisation des données. Si les erreurs de cache sont importantes et que l'utilisation de la mémoire est faible, augmentez la durée TTL pour augmenter le taux d'accès au cache.

Tenez compte des exigences commerciales et de la conformité

Les politiques de conservation des données peuvent dicter la durée TTL maximale que vous pouvez définir pour les informations sensibles ou personnelles.

Comportement du cache si vous définissez TTL sur zéro

Si vous définissez TTL sur 0, le cache d'éléments et le cache de requêtes présentent les comportements suivants :

- Cache d'éléments : les éléments du cache sont actualisés uniquement en cas d'éviction du LRU ou d'une opération d'écriture simultanée.
- Cache de requêtes : les réponses aux requêtes ne sont pas mises en cache.

Mise en cache de plusieurs tables avec un cluster DAX

Pour les charges de travail comportant plusieurs petites tables DynamoDB qui ne nécessitent pas de caches individuels, un seul cluster DAX met en cache les demandes relatives à ces tables. Cela permet une utilisation plus flexible et plus efficace de DAX, en particulier pour les applications qui accèdent à plusieurs tables et nécessitent des lectures hautes performances.

À l'instar du APIs plan de données DynamoDB, les requêtes DAX nécessitent un nom de table. Si vous utilisez plusieurs tables dans le même cluster DAX, aucune configuration spécifique n'est nécessaire. Cependant, vous devez vous assurer que les autorisations de sécurité du cluster autorisent l'accès à toutes les tables mises en cache.

Considérations relatives à l'utilisation de DAX avec plusieurs tables

Lorsque vous utilisez DAX avec plusieurs tables DynamoDB, vous devez tenir compte des points suivants :

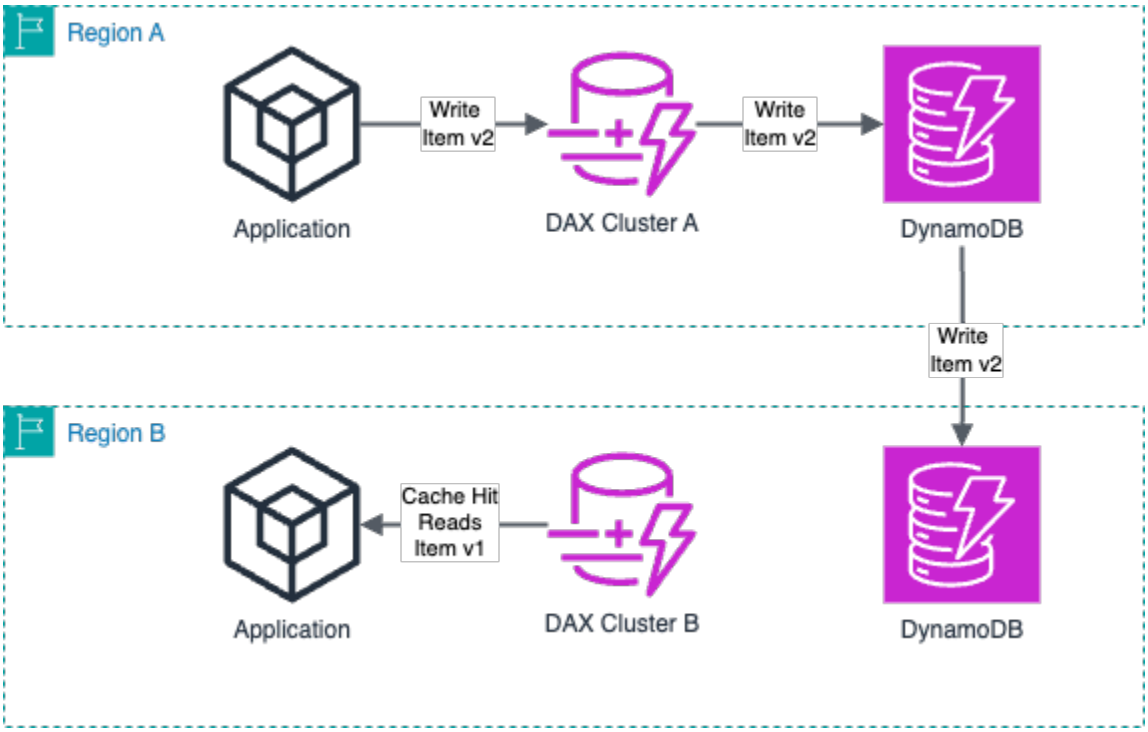
- Gestion de la mémoire : lorsque vous utilisez DAX avec plusieurs tables, vous devez tenir compte de la taille totale de votre jeu de données de travail. Toutes les tables de votre jeu de données partageront le même espace mémoire que le type de nœud que vous avez sélectionné.
- Allocation de ressources : les ressources du cluster DAX sont partagées entre toutes les tables mises en cache. Cependant, une table à trafic élevé peut entraîner l'éviction des données des tables plus petites voisines.
- Économies d'échelle : regroupez les petites ressources dans un cluster DAX plus grand afin de répartir le trafic sortant selon un schéma plus stable. Pour le nombre total de ressources de lecture dont le cluster DAX a besoin, il est également économique de disposer de trois nœuds ou plus. Cela augmente également la disponibilité de toutes les tables mises en cache dans le cluster.

Réplication des données dans les tables globales DAX et DynamoDB

DAX est un service basé sur une région, de sorte qu'un cluster ne connaît que le trafic au sein de sa Région AWS. Les tables globales contournent le cache lorsqu'elles répliquent des données d'une autre région.

Avec une durée de TTL plus longue, les données périmées peuvent rester plus longtemps dans votre région secondaire que dans la région principale. Cela peut entraîner des erreurs de cache dans le cache local de la région secondaire.

Le schéma suivant montre la réplication des données qui se produit au niveau de la table globale dans la région source A. Le cluster DAX de la région B ne connaît pas immédiatement les données récemment répliquées à partir de la région source A.



Disponibilité de région DAX

Les régions qui prennent en charge les tables DynamoDB ne prennent pas toutes en charge le déploiement de clusters DAX. Si votre application nécessite une faible latence de lecture via DAX, consultez d'abord la liste des régions qui prennent en charge DAX. Sélectionnez ensuite une région pour votre table DynamoDB.

Comportement de mise en cache DAX

DAX effectue une mise en cache des métadonnées et négative. La compréhension de ces comportements de mise en cache vous aidera à gérer efficacement les métadonnées d'attribut des éléments mis en cache et des entrées de cache négatives.

- Mise en cache des métadonnées : les clusters DAX conservent indéfiniment les métadonnées sur les noms d'attribut des éléments mis en cache. Ces métadonnées sont conservées même après l'expiration de l'élément ou son éviction du cache.

  Au fil du temps, les applications qui utilisent un nombre illimité de noms d'attributs peuvent entraîner un épuisement de la mémoire du cluster DAX. Cette limitation s'applique uniquement aux noms d'attributs de niveau supérieur, mais pas aux noms d'attributs imbriqués. Les exemples de noms d'attributs illimités incluent les horodatages et les sessions. UUIDs IDs Bien que vous puissiez utiliser les horodatages et les sessions IDs comme valeurs d'attribut, nous vous recommandons d'utiliser des noms d'attributs plus courts et plus prévisibles.
- Mise en cache négative : si une erreur de cache se produit et que la lecture d'une table DynamoDB ne produit aucun élément correspondant, DAX ajoute une entrée de cache négative dans le cache d'éléments ou de requêtes correspondant. Cette entrée est conservée jusqu'à ce que la durée TTL du cache expire ou qu'une écriture simultanée soit effectuée. DAX continue de renvoyer cette entrée de cache négative pour les demandes futures.

  Si le comportement de mise en cache négatif ne correspond pas au modèle de votre application, lisez la table DynamoDB directement lorsque DAX renvoie un résultat vide. Nous vous recommandons également de définir une durée de cache TTL inférieure afin d'éviter des résultats vides persistants dans le cache et d'améliorer la cohérence avec la table.

Dimensionnement de votre cluster DAX

La capacité et la disponibilité totales d'un cluster DAX dépendent du type et du nombre de nœuds. Un plus grand nombre de nœuds dans le cluster augmente sa capacité de lecture, mais pas sa capacité d'écriture. Les types de nœuds plus importants (jusqu'à r5.8xlarge) peuvent gérer un plus grand nombre d'écritures, mais si le nombre de nœuds est insuffisant, cela peut avoir un impact sur la disponibilité en cas de défaillance d'un nœud. Pour plus d'informations sur le dimensionnement de votre cluster DAX, consultez le Guide de dimensionnement de cluster DAX.

Les sections suivantes traitent des différents aspects du dimensionnement que vous devez prendre en compte pour équilibrer le type et le nombre de nœuds afin de créer un cluster évolutif et rentable.

Dans cette section

- Planification de la disponibilité
- Planning du débit d'écriture
- Planification du débit de lecture

- Planification de la taille du jeu de données
- Calcul approximatif des exigences de capacité du cluster
- Approximation de la capacité de débit du cluster par type de nœud
- Mise à l'échelle de la capacité d'écriture dans les clusters DAX

Planification de la disponibilité

Lorsque vous dimensionnez un cluster DAX, vous devez d'abord vous concentrer sur sa disponibilité ciblée. La disponibilité d'un service en cluster, tel que DAX, est une dimension du nombre total de nœuds du cluster. Étant donné qu'un cluster à nœud unique ne tolère aucune défaillance, sa disponibilité est égale à un nœud. Dans un cluster à 10 nœuds, la perte d'un seul nœud a un impact minimal sur la capacité globale du cluster. Cette perte n'a pas d'impact direct sur la disponibilité, car les nœuds restants peuvent toujours répondre aux demandes de lecture. Pour reprendre les écritures, DAX désigne rapidement un nouveau nœud primaire.

DAX est basé sur le VPC. Il utilise un groupe de sous-réseaux pour déterminer les zones de disponibilité dans lesquelles il peut exécuter des nœuds et les adresses IP à utiliser à partir des sous-réseaux. Pour les charges de travail de production, nous vous recommandons vivement d'utiliser DAX avec au moins trois nœuds dans des zones de disponibilité différentes. Cela garantit que le cluster dispose de plus d'un nœud pour traiter les demandes, même en cas de défaillance d'un seul nœud ou d'une seule zone de disponibilité. Un cluster peut comporter jusqu'à 11 nœuds, dont l'un est un nœud primaire et 10 sont des réplicas en lecture.

Planning du débit d'écriture

Tous les clusters DAX disposent d'un nœud primaire pour les demandes en écriture simultanée. La taille du type de nœud du cluster détermine sa capacité d'écriture. L'ajout de réplicas en lecture supplémentaires n'augmente pas la capacité d'écriture du cluster. Par conséquent, vous devez tenir compte de la capacité d'écriture lors de la création du cluster, car vous ne pourrez pas modifier le type de nœud ultérieurement.

Si votre application doit utiliser DAX en écriture simultanée pour mettre à jour le cache des éléments, envisagez d'utiliser davantage les ressources du cluster pour faciliter l'écriture. Les écritures sur DAX consomment environ 25 fois plus de ressources que les lectures en cache. Cela peut nécessiter un type de nœud plus important que pour les clusters en lecture seule.

Pour obtenir des conseils supplémentaires sur la manière de déterminer si la version à écriture simultanée ou non allouée convient le mieux à votre application, consultez Politiques pour les écritures.

Planification du débit de lecture

La capacité de lecture d'un cluster DAX dépend du taux de réussite du cache de votre charge de travail. Étant donné que DAX lit les données de DynamoDB en cas d'échec du cache, il consomme environ 10 fois plus de ressources de cluster qu'un accès au cache. Pour augmenter le nombre d'accès au cache, augmentez le paramètre TTL du cache afin de définir la période pendant laquelle un élément est stocké dans le cache. Une durée TTL supérieure augmente toutefois les chances de lire les anciennes versions des éléments, sauf si les mises à jour sont écrites via DAX.

Pour vous assurer que le cluster dispose d'une capacité de lecture suffisante, mettez-le à l'échelle horizontalement comme indiqué dans Mise à l'échelle horizontale d'un cluster. L'ajout de nœuds supplémentaires ajoute des réplicas de lecture au pool de ressources, tandis que la suppression de nœuds réduit la capacité de lecture. Lorsque vous sélectionnez le nombre de nœuds et leur taille pour un cluster, tenez compte de la capacité de lecture minimale et maximale requise. Si vous ne pouvez pas mettre à l'échelle horizontalement un cluster avec des types de nœuds plus petits pour répondre à vos exigences de lecture, utilisez un type de nœud plus grand.

Planification de la taille du jeu de données

Chaque type de nœud disponible possède une taille de mémoire définie pour que DAX puisse mettre en cache les données. Si le type de nœud est trop petit, l'ensemble de données de travail demandé par une application sera trop important pour la mémoire et entraînera des erreurs de cache. Étant donné que les nœuds de grande taille prennent en charge des caches de plus grande taille, utilisez un type de nœud supérieur au jeu de données estimé que vous devez mettre en cache. Un cache plus important améliore également le taux d'accès au cache.

Vous pouvez obtenir des retours décroissants pour la mise en cache d'éléments après quelques lectures répétées. Calculez la taille de la mémoire pour les éléments fréquemment consultés et assurez-vous que le cache est suffisamment grand pour stocker ce jeu de données.

Calcul approximatif des exigences de capacité du cluster

Vous pouvez estimer les besoins en capacité totale de votre charge de travail pour vous aider à sélectionner la taille et le nombre appropriés de nœuds de cluster. Pour effectuer cette estimation, calculez la variable demande normalisée par seconde (RPS normalisé). Cette variable représente le

nombre total d'unités de travail que votre application doit prendre en charge avec le cluster DAX, y compris les accès au cache, les erreurs de cache et les écritures. Pour calculer le RPS normalisé, utilisez les entrées suivantes :

- `ReadRPS_CacheHit` : spécifie le nombre de lectures par seconde qui entraînent un accès au cache.
- `ReadRPS_CacheMiss` : spécifie le nombre de lectures par seconde qui entraînent une erreur de cache.
- `WriteRPS` : spécifie le nombre d'écritures par seconde qui passeront par DAX.
- `DaxNodeCount` : spécifie le nombre de nœuds dans le cluster DAX.
- `Size` : spécifie la taille de l'élément en cours d'écriture ou de lecture en Ko, arrondie au Ko le plus proche.
- `10x_ReadMissFactor` : représente une valeur de 10. En cas d'échec du cache, DAX utilise environ 10 fois plus de ressources que les accès au cache.
- `25x_WriteFactor` : représente une valeur de 25, car une écriture simultanée DAX utilise environ 25 fois plus de ressources que les accès au cache.

Vous pouvez calculer le RPS normalisé à l'aide de la formule suivante.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Prenons l'exemple des valeurs d'entrée suivantes :

- `ReadRPS_CacheHit` = 50 000
- `ReadRPS_CacheMiss` = 1 000
- `ReadMissFactor` = 1
- `Size` = 2 Ko
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

En remplaçant ces valeurs dans la formule, vous pouvez calculer le RPS normalisé comme suit.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

Dans cet exemple, la valeur calculée du RPS normalisé est 135 000. Cependant, cette valeur RPS normalisée ne tient pas compte du maintien de l'utilisation du cluster en dessous de 100 % ou de la perte de nœuds. Nous vous recommandons de prendre en compte la capacité supplémentaire. Pour ce faire, déterminez le plus élevé des deux facteurs multiplicateurs : l'utilisation cible ou la tolérance à la perte de nœuds. Multipliez ensuite le RPS normalisé par le facteur le plus élevé pour obtenir une demande cible par seconde (RPS cible).

- Taux d'utilisation cible

  Étant donné que les impacts sur les performances augmentent les erreurs de cache, nous ne recommandons pas d'exécuter le cluster DAX pour 100 % des utilisations. Idéalement, vous devez maintenir le taux d'utilisation du cluster à 70 % ou moins. Pour ce faire, multipliez le RPS normalisé par 1,43.

- Tolérance à la perte de nœuds

  En cas de défaillance d'un nœud, votre application doit être en mesure de répartir ses demandes entre les nœuds restants. Pour vous assurer que le taux d'utilisation des nœuds reste inférieur à 100 %, choisissez un type de nœud suffisamment grand pour absorber le trafic supplémentaire jusqu'à ce que le nœud défaillant soit remis en ligne. Pour un cluster comportant moins de nœuds, chaque nœud doit tolérer des augmentations de trafic supérieures en cas de défaillance d'un nœud.

  En cas d'échec du nœud primaire, DAX bascule automatiquement vers un réplica en lecture et le désigne en tant que nouveau nœud principal. En cas d'échec d'un nœud réplica, d'autres nœuds dans le cluster DAX peuvent encore servir les demandes jusqu'à la récupération du nœud en échec.

  Par exemple, un cluster DAX à 3 nœuds présentant une défaillance de nœud nécessite une capacité supplémentaire de 50 % sur les deux nœuds restants. Cela nécessite un facteur multiplicateur de 1,5. À l'inverse, un cluster de 11 nœuds dont un nœud est défaillant nécessite une capacité supplémentaire de 10 % sur les nœuds restants, soit un facteur multiplicateur de 1,1.

Vous pouvez calculer le RPS cible à l'aide de la formule suivante.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Prenons l'exemple des valeurs suivantes pour calculer le RPS cible :

- `Normalized RPS` = 135 000
- `TargetUtilization` = 1,43

  Parce que nous visons une utilisation maximale du cluster de 70 %, nous avons défini `TargetUtilization` sur 1,43.
- `NodeLossTolerance` = 1,5

  Supposons que nous utilisions un cluster à 3 nœuds. Par conséquent, nous définissons `NodeLossTolerance` sur 50 %.

En remplaçant ces valeurs dans la formule, vous pouvez calculer le RPS cible comme suit.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

Dans cet exemple, étant donné que la valeur de `NodeLossTolerance` est supérieure à `TargetUtilization`, nous calculons la valeur de RPS cible avec `NodeLossTolerance`. Cela nous donne un RPS cible de 202 500, soit la capacité totale que le cluster DAX doit prendre en charge. Pour déterminer le nombre de nœuds dont vous aurez besoin dans un cluster, mappez le RPS cible à une colonne appropriée dans le tableau suivant. Pour cet exemple de RPS cible de 202 500, vous avez besoin du type de nœud dax.r5.large avec trois nœuds.

Approximation de la capacité de débit du cluster par type de nœud

À l'aide de la Target RPS formula, vous pouvez estimer la capacité du cluster pour différents types de nœuds. Le tableau suivant indique les capacités approximatives des clusters de 1, 3, 5 et 11 types de nœuds. Ces capacités ne remplacent pas la nécessité de tester la charge de DAX avec vos propres données et modèles de demandes. De plus, ces capacités n'incluent pas les instances de type t en raison de leur manque de capacité CPU fixe. L'unité pour toutes les valeurs du tableau suivant est le RPS normalisé.

Type de nœud (mémoire)	1 nœud	3 nœuds	5 nœuds	11 nœuds
dax.r5.24xlarge (768 Go)	1 M	3 M	5 M	11 M
dax.r5.16xlarge (512 Go)	1 M	3 M	5 M	11 M
dax.r5.12xlarge (384 Go)	1 M	3 M	5 M	11 M
dax.r5.8xlarge (256 Go)	1 M	3 M	5 M	11 M
dax.r5.4xlarge (128 Go)	600 000	1,8 M	3 M	6,6 M
dax.r5.2xlarge (64 Go)	300 000	900 000	1,5 M	3,3 M
dax.r5.xlarge (32 Go)	150 000	450 000	750 000	1,65 M
dax.r5.large (16 Go)	75 000	225 000	375 000	825 000

En raison de la limite maximale de 1 million de NPS (opérations réseau par seconde) pour chaque nœud, les nœuds de type dax.r5.8xlarge ou supérieur ne contribuent pas à augmenter la capacité du cluster. Les types de nœuds supérieurs à 8xlarge peuvent ne pas contribuer à la capacité de débit totale du cluster. Cependant, ces types de nœuds peuvent être utiles pour stocker un jeu de données de travail plus important en mémoire.

Mise à l'échelle de la capacité d'écriture dans les clusters DAX

Chaque écriture sur DAX consomme 25 requêtes normalisées sur chaque nœud. Comme il existe une limite d'un million de RPS pour chaque nœud, un cluster DAX est limité à 40 000 écritures par seconde, sans tenir compte de l'utilisation en lecture.

Si votre cas d'utilisation nécessite plus de 40 000 écritures par seconde dans le cache, vous devez utiliser des clusters DAX distincts et partitionner les écritures entre eux. Comme dans DynamoDB, vous pouvez hacher la clé de partition pour les données que vous écrivez dans le cache. Utilisez ensuite le modulo pour déterminer dans quelle partition écrire les données.

L'exemple suivant calcule le hachage d'une chaîne d'entrée. Il calcule ensuite le module de la valeur de hachage avec 10.

```
def hash_modulo(input_string):
    # Compute the hash of the input string
    hash_value = hash(input_string)

    # Compute the modulus of the hash value with 10
    bucket_number = hash_value % 10

    return bucket_number

#Example usage
if _name_ == "_main_":
    input_string = input("Enter a string: ")
    result = hash_modulo(input_string)
    print(f"The hash modulo 10 of '{input_string}' is: {result}.")
```

Déploiement d'un cluster

La création d'un nouveau cluster DAX nécessite des configurations allant au-delà de celles requises pour DynamoDB. Ces configurations sont particulièrement destinées à la mise en réseau, car DAX est basé sur Amazon VPC. Cela vous offre un contrôle total sur votre environnement réseau virtuel, y compris le placement des ressources, la connectivité et la sécurité. Cette section présente les bonnes pratiques relatives aux paramètres nécessaires lors de la création du cluster.

Pour plus d'informations sur le choix des nœuds de cluster, consultez Dimensionnement de votre cluster DAX.

Dans cette section

- Configuration de réseaux
- Configuration de la sécurité
- Groupe de paramètres
- Fenêtre de maintenance

Configuration de réseaux

DAX utilise un groupe de sous-réseaux pour déterminer les zones de disponibilité dans lesquelles il peut exécuter des nœuds et les adresses IP à utiliser à partir des sous-réseaux. Pour minimiser le temps de latence entre votre application et DAX, les sous-réseaux et les zones de disponibilité de vos serveurs d'applications et du cluster DAX doivent être identiques.

Nous vous recommandons de répartir les nœuds DAX entre plusieurs zones de disponibilité. L'option par défaut, Allocation automatique, s'en charge pour vous.

Pour connaître les bonnes pratiques relatives à la configuration de votre VPC, consultez Commencer avec Amazon VPC dans le Guide de l'utilisateur Amazon VPC.

Configuration de la sécurité

Cette section décrit les mesures de sécurité que vous devez mettre en œuvre pour vos applications utilisant DAX. Cette section décrit également brièvement le support inclus dans DAX pour le chiffrement des données.

IAM

DAX et DynamoDB ont des mécanismes de contrôle d'accès distincts. DAX nécessite un rôle IAM pour accéder à vos tables DynamoDB. Ce rôle doit respecter le principe du moindre privilège et autoriser l'accès uniquement à des tables et à des opérations DynamoDB spécifiques, telles que et. GetItemPutItem Pour plus d'informations sur les mécanismes de contrôle d'accès fournis par DAX, consultez Contrôle d'accès à DAX.

Chiffrement

Vous configurez le chiffrement au repos et le chiffrement en transit lors de la création d'un cluster DAX. Ils sont activés par défaut. Nous vous recommandons de conserver les paramètres de chiffrement par défaut, sauf si les exigences de l'entreprise l'empêchent. Pour plus d'informations, consultez Chiffrement au repos DAX et Chiffrement DAX en transit.

Groupe de paramètres

DAX applique un ensemble de configurations sur chaque nœud d'un cluster appelé groupe de paramètres. Vous pouvez modifier cette configuration après avoir créé le cluster.

Le groupe de paramètres DAX contient les paramètres TTL pour le cache d'élément et le cache de requête. La durée TTL par défaut est de 5 minutes. Vous pouvez remplacer la durée TTL par une valeur entière supérieure ou égale à 1 milliseconde.

Vous ne pouvez pas modifier les groupes de paramètres lorsqu'une instance DAX en cours d'exécution les utilise. Vous pouvez modifier les valeurs des groupes de paramètres pendant la durée d'indisponibilité d'un cluster DAX.

Fenêtre de maintenance

Une fenêtre de maintenance hebdomadaire est configurée pour le cluster DAX pour permettre d'effectuer des mises à niveau logicielles et des correctifs occasionnels sur vos nœuds. Au cours de cette fenêtre, DAX effectue des mises à jour continues des nœuds. Les clusters comportant plusieurs nœuds ne perdent pas leur disponibilité lors de ces mises à jour, mais leur capacité est réduite jusqu'au retour du nœud. Si votre organisation connaît une période prévisible de faible utilisation, nous vous conseillons de définir manuellement la fenêtre de maintenance sur cette période.

Gestion des opérations d'un cluster

DAX gère la maintenance et l'état du cluster pour vous. Cependant, vous devez fournir des données opérationnelles pour mettre à l'échelle le cluster horizontalement ou verticalement en fonction de vos habitudes d'utilisation. Cette section décrit le processus recommandé pour mettre à l'échelle vos clusters DAX.

Dans cette section

- Mise à l'échelle horizontale d'un cluster
- Mise à l'échelle verticale d'un cluster

Mise à l'échelle horizontale d'un cluster

La mise à l'échelle d'un cluster DAX implique d'ajuster sa capacité pour répondre aux demandes de débit. Cet ajustement s'effectue en augmentant ou en diminuant le nombre de nœuds (réplicas) dans le cluster pendant son exécution. Ce processus, connu sous le nom de mise à l'échelle horizontale, permet de répartir la charge de travail sur un plus grand nombre de nœuds ou de la consolider sur un nombre réduit de nœuds lorsque la demande est faible.

Vous pouvez augmenter ou réduire horizontalement votre cluster DAX à l'aide des commandes `decrease-replication-factor` ou `increase-replication-factor` de l'AWS CLI.

Augmentation du facteur de réplication (augmentation horizontale)

L'augmentation du facteur de réplication d'un cluster DAX ajoute des nœuds supplémentaires au cluster. L'exemple suivant illustre l'utilisation de la commande `increase-replication-factor`.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Dans cette commande, l'argument `cluster-name` indique le nom de votre cluster. Par exemple, *yourClusterName*.
- L'argument `new-replication-factor` spécifie le nombre total de nœuds à ajouter dans le cluster après la mise à l'échelle. Cela inclut le nœud primaire et les nœuds de réplica. Par exemple, si votre cluster compte actuellement 3 nœuds et que vous souhaitez en ajouter 2 autres, définissez la valeur de `new-replication-factor` sur 5.

Diminution du facteur de réplication (réduction horizontale)

La réduction du facteur de réplication d'un cluster DAX supprime des nœuds du cluster. La suppression de nœuds peut contribuer à réduire les coûts pendant les périodes de faible demande. L'exemple suivant illustre l'utilisation de la commande `decrease-replication-factor`.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Dans cette commande, l'argument `cluster-name` indique le nom de votre cluster. Par exemple, *yourClusterName*.
- L'argument `new-replication-factor` spécifie le nombre réduit de nœuds dans votre cluster après la mise à l'échelle. Ce nombre doit être inférieur au facteur de réplication actuel et doit inclure le nœud primaire. Par exemple, si votre cluster compte 5 nœuds et que vous souhaitez en supprimer 2, définissez la valeur de `new-replication-factor` sur 3.

Considérations relatives à la mise à l'échelle horizontale

Lorsque vous planifiez une mise à l'échelle horizontale, tenez compte des éléments suivants :

- Nœud primaire : le cluster DAX inclut un nœud primaire. Le facteur de réplication inclut ce nœud primaire. Par exemple, un facteur de réplication de 3 signifie un nœud primaire et deux nœuds de réplica.

- Disponibilité : l'ajout ou la suppression de nœuds DAX modifie la disponibilité et la tolérance aux pannes du cluster. Un nombre supérieur de nœuds peut améliorer la disponibilité, mais également augmenter les coûts.
- Migration des données : lorsque vous augmentez le facteur de réplication, DAX gère automatiquement la distribution des données sur le nouveau jeu de nœuds. Lorsqu'un nouveau nœud commence à traiter du trafic, son cache est déjà réchauffé. Toutefois, au cours de ce processus, il peut y avoir un impact temporaire sur les performances lors de la migration des données.

Assurez-vous de surveiller de près vos clusters DAX pendant et après le processus de dimensionnement afin de vous assurer qu'ils fonctionnent comme prévu, puis apportez des ajustements supplémentaires si nécessaire.

Mise à l'échelle verticale d'un cluster

Pour mettre à l'échelle verticalement la taille des nœuds d'un cluster existant, vous devez créer un cluster et migrer le trafic des applications vers celui-ci. La migration vers un nouveau cluster avec différents nœuds implique plusieurs étapes afin de garantir une transition fluide, avec un impact minimal sur les performances et la disponibilité de votre application.

Pour créer un cluster permettant de mettre à l'échelle verticalement la taille de votre nœud, tenez compte des points suivants :

- Accédez à votre configuration actuelle : passez en revue les métriques de votre cluster DAX actuel pour déterminer la taille et la quantité dont vous avez besoin pour les nouveaux nœuds. Appuyez-vous sur ces informations pour définir la taille de votre cluster. Pour plus d'informations, consultez Dimensionnement de votre cluster DAX.
- Configurez un nouveau cluster DAX : créez un nouveau cluster DAX avec le type et la quantité de nœuds que vous avez déterminés. Vous pouvez utiliser les paramètres de configuration existants de votre groupe de paramètres, sauf si vous devez effectuer des ajustements.
- Synchronisez les données : DAX étant une couche de mise en cache pour DynamoDB, il n'est pas nécessaire de migrer les données directement. Cependant, le nouveau cluster DAX ne conservera aucun de vos jeux de données de travail en mémoire tant que vous ne lui aurez pas envoyé du trafic.
- Mettez à jour la configuration de l'application : mettez à jour la configuration de votre application pour qu'elle pointe vers le point de terminaison du nouveau cluster DAX. Vous devrez peut-être

modifier le code ou mettre à jour les variables d'environnement, en fonction de la configuration de votre application.

Pour réduire l'impact lorsque vous passez à un nouveau cluster, envoyez du trafic canary vers le nouveau cluster à partir d'une petite partie de votre flotte d'applications. Vous pouvez le faire en déployant lentement les mises à jour des applications ou en utilisant une entrée DNS de routage basée sur le poids devant votre point de terminaison DAX.

- Surveillez et optimisez : une fois que vous êtes passé au nouveau cluster DAX, surveillez de près ses métriques de performance et ses journaux pour détecter tout problème. Soyez prêt à ajuster le nombre de nœuds en fonction des modèles de charge de travail mis à jour.

  Tant que le nouveau cluster ne met pas correctement en cache votre jeu de données de travail, vous constaterez des taux d'échec du cache et des latences supérieurs.
- Mise hors service de l'ancien cluster : lorsque vous êtes certain que le nouveau cluster fonctionne comme prévu, vous pouvez mettre hors service l'ancien cluster DAX en toute sécurité afin d'éviter des coûts inutiles.

Surveillance de DAX

Vous pouvez surveiller les métriques clés, comme le taux d'accès au cache, afin de garantir des performances optimales du cluster DAX, de diagnostiquer les problèmes et de déterminer à quel moment vous devez mettre à l'échelle le cluster. La vérification régulière des métriques clés vous aide à maintenir les performances, la stabilité et la rentabilité en mettant à l'échelle le cluster selon les exigences de votre charge de travail. Pour plus d'informations sur la surveillance de DAX, consultez Surveillance en production.

La liste suivante présente certaines des métriques clés que vous devez surveiller :

- Taux d'accès au cache : indique l'efficacité avec laquelle DAX traite les données mises en cache, réduisant ainsi le besoin d'accéder aux tables DynamoDB sous-jacentes. Quelques erreurs de cache pour le cluster indiquent une bonne efficacité de la mise en cache. Cependant, un faible nombre d'accès au cache suggère que vous devrez peut-être revoir le paramètre TTL de mise en cache ou que la charge de travail ne convient pas à la mise en cache.

  Utilisez Amazon CloudWatch pour calculer le taux d'accès au cache de votre cluster DAX. Comparez les métriques `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits` et `QueryCacheMisses` pour obtenir ce ratio. La formule suivante indique comment le taux d'accès

au cache est calculé. Pour calculer le ratio à l'aide de cette formule, divisez le nombre d'accès au cache par la somme des accès et des échecs.

```
Cache hit ratio = Cache hits / (Cache hits + Cache misses)
```

Le taux d'accès du cache est un nombre compris entre 0 et 1, qui est représenté en pourcentage. Un pourcentage supérieur indique une meilleure utilisation globale du cache.

- ErrorRequestCount— Nombre de demandes ayant entraîné des erreurs utilisateur signalées par le nœud ou le cluster. `ErrorRequestCount`inclut les demandes qui ont été limitées par le nœud ou le cluster. La surveillance des erreurs des utilisateurs peut vous aider à identifier les erreurs de dimensionnement ou item/partition les modèles récurrents dans votre application.
- Latences opérationnelles : la surveillance de la latence des opérations de lecture et d'écriture vers et depuis le cluster DAX peut vous aider à identifier les goulots d'étranglement liés aux performances. L'augmentation des latences peut indiquer des problèmes liés à la configuration de votre cluster DAX, au réseau ou à la nécessité d'une mise à l'échelle.
- Consommation réseau : surveillez les métriques `NetworkBytesIn` et `NetworkBytesOut` pour surveiller le trafic réseau de votre cluster DAX. Une augmentation inattendue du débit du réseau peut entraîner une augmentation du nombre de demandes des clients ou des modèles de requêtes inefficaces entraînant le transfert d'une plus grande quantité de données.

  La surveillance de la consommation du réseau vous aide à gérer les coûts de votre cluster DAX. Cela garantit également que le réseau ne devient pas un goulot d'étranglement pour les performances du cluster.
- Taux d'éviction : indique la fréquence à laquelle des éléments sont retirés de votre cache pour faire de la place à de nouveaux éléments. Si le taux d'éviction augmente au fil du temps, votre cache est peut-être trop petit ou votre stratégie de mise en cache n'est pas efficace.

  Surveillez la `EvictedSize` métrique CloudWatch pour déterminer si la taille de votre cache est adaptée à votre charge de travail. Si la taille totale faisant l'objet d'une éviction ne cesse de croître, vous devrez peut-être augmenter verticalement votre cluster DAX pour qu'il puisse accueillir un cache plus important.
- Utilisation du CPU : désigne le pourcentage d'utilisation du CPU du nœud ou du cluster. Il s'agit d'une métrique essentielle à surveiller pour n'importe quelle base de données ou système de mise en cache. Une utilisation élevée du CPU peut indiquer que votre cluster DAX est peut-être surchargé et doit être mis à l'échelle pour faire face à la demande accrue.

Surveillez la métrique `CPUUtilization` de votre cluster DAX. Si l'utilisation de votre CPU approche ou dépasse régulièrement 70 à 80 %, pensez à augmenter verticalement votre cluster DAX comme décrit dans la section suivante.

Si le nombre de demandes envoyées à DAX dépasse la capacité d'un nœud, DAX limite le taux d'acceptation des demandes supplémentaires. Pour ce faire, il renvoie un ThrottlingException. DAX évalue en continu l'utilisation du CPU de votre cluster pour déterminer le volume de demandes qu'il peut traiter tout en conservant un état de cluster sain.

Vous pouvez surveiller la `ThrottledRequestCount` métrique sur laquelle DAX publie. CloudWatch Si ces exceptions s'affichent régulièrement, vous devez envisager de mettre à l'échelle votre cluster.

Mise à l'échelle de votre cluster DAX à l'aide des données de surveillance

Vous pouvez déterminer si vous devez augmenter ou réduire verticalement votre cluster DAX en surveillant ses indicateurs de performance.

- Augmentation verticale ou horizontale : si votre cluster DAX utilise beaucoup le processeur, présente de faibles taux d'accès au cache (après optimisation de la stratégie de mise en cache) ou des latences de fonctionnement élevées, vous devez augmenter verticalement votre cluster. L'ajout de nœuds supplémentaires, également appelé augmentation horizontale, peut aider à répartir la charge de manière plus uniforme. Pour les charges de travail impliquant une augmentation du nombre d'écritures par seconde, vous devrez peut-être choisir des nœuds plus puissants (augmentation verticale).
- Réduction verticale : si vous constatez régulièrement une faible utilisation du CPU et des latences de fonctionnement inférieures à vos seuils, vous avez peut-être surprovisionné les ressources. Dans ce cas, réduisez verticalement le nombre de nœuds pour réduire les coûts. Vous pouvez réduire le nombre de nœuds à 1 pendant les périodes de faible utilisation, mais vous ne pouvez pas arrêter complètement le cluster.

Utilisation de DynamoDB avec d'autres services AWS

Amazon DynamoDB est intégré àAWS d'autres services, ce qui vous permet d'automatiser les tâches répétitives ou de créer des applications couvrant plusieurs services.

Rubriques

- Configuration des AWS informations d'identification à l'aide d'Amazon Cognito pour DynamoDB
- Intégration avec Amazon Redshift
- Traitement de données DynamoDB avec Apache Hive sur Amazon EMR
- Intégration de DynamoDB à Amazon S3
- Intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse
- Intégration zéro ETL de DynamoDB à Amazon OpenSearch Service
- Intégration de DynamoDB à Amazon EventBridge
- Intégration de DynamoDB à Amazon Managed Streaming for Apache Kafka
- Bonnes pratiques relatives à l’intégration à DynamoDB

Configuration des AWS informations d'identification à l'aide d'Amazon Cognito pour DynamoDB

La méthode recommandée pour obtenir des AWS informations d'identification pour vos applications Web et mobiles est d'utiliser Amazon Cognito. Amazon Cognito vous permet d'éviter de coder en dur vos AWS informations d'identification sur vos fichiers. Il utilise des rôles Gestion des identités et des accès AWS(IAM) pour générer des informations d'identification temporaires pour les utilisateurs authentifiés et non authentifiés de votre application.

Par exemple, pour configurer vos JavaScript fichiers afin qu'ils utilisent un rôle non authentifié Amazon Cognito pour accéder au service Web Amazon DynamoDB, procédez comme suit.

Pour configurer les informations d’identification à intégrer avec Amazon Cognito

1. Créez un groupe d’identités Amazon Cognito qui autorise les identités non authentifiées.

```
aws cognito-identity create-identity-pool \
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
```

```
    --output json
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. Copiez la stratégie suivante dans un fichier nommé `myCognitoPolicy.json`. Remplacez l'ID du pool d'identités (*`us-west-2:12345678-1ab2-123a-1234-a12345ab12`*) par le vôtre `IdentityPoolId` obtenu à l'étape précédente.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. Créez un rôle IAM qui endosse la politique précédente. Ainsi, Amazon Cognito devient une entité de confiance capable d'endosser le rôle `Cognito_DynamoPoolUnauth`.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. Octroyez au rôle `Cognito_DynamoPoolUnauth` un accès total à DynamoDB en attachant une politique gérée (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> Sinon, vous pouvez octroyer un accès précis à DynamoDB. Pour plus d'informations, consultez Utilisation de conditions de politique IAM pour un contrôle d'accès précis.

5. Obtenez et copiez le rôle IAM Amazon Resource Name (ARN).

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Ajoutez le rôle `Cognito_DynamoPoolUnauth` au groupe d'identités `DynamoPool`. Le format à spécifier est `KeyName=string`, `KeyName` étant `unauthenticated`, et la chaîne correspond à l'ARN de rôle obtenu au cours de l'étape précédente.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. Spécifiez les informations d'identification Amazon Cognito dans vos fichiers. Modifiez `IdentityPoolId` et `RoleArn` en conséquence.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Vous pouvez désormais exécuter vos JavaScript programmes sur le service Web DynamoDB à l'aide des informations d'identification Amazon Cognito. Pour plus d'informations, consultez Définition des informations d'identification dans un navigateur web dans le Guide de démarrage AWS SDK pour JavaScript.

Intégration avec Amazon Redshift

Amazon Redshift est un service d'entrepôt de données rapide, entièrement géré et doté d'une capacité de plusieurs pétaoctets. Il permet d'analyser de manière efficace, simple et économique toutes vos données grâce à vos outils d'informatique décisionnelle existants.

DynamoDB et Amazon Redshift peuvent être utilisés ensemble pour répondre aux différents besoins de stockage et de traitement des données au sein d'une application ou d'un écosystème de données.

Consultez les rubriques ci-dessous pour plus d'informations sur l'intégration de DynamoDB à Amazon Redshift.

Rubriques

- Considérations relatives à l'intégration entre comptes avec CMK
- Intégration zéro ETL de DynamoDB à Amazon Redshift
- Chargement de données de DynamoDB dans Amazon Redshift à l'aide de la commande COPY

Considérations relatives à l'intégration entre comptes avec CMK

Lorsque vous tentez d'intégrer DynamoDB à Amazon Redshift, l'action initiale est lancée depuis Amazon Redshift. Sans les autorisations appropriées, cette action peut entraîner un échec silencieux. Les sections suivantes détaillent les autorisations requises pour cette intégration entre comptes.

AWS KMS Politiques et autorisations requises

Dans les examples, remplacez les espaces réservés suivants :

- `111122223333`:Compte AWS ID où Amazon Redshift est hébergé
- `444455556666`:Compte AWS ID où DynamoDB est hébergé
- `REDSHIFT_ROLE_NAME` : nom de rôle IAM utilisé par Amazon Redshift
- `REGION`: L'Région AWS endroit où se trouvent vos ressources
- `TABLE_NAME` : nom de votre table DynamoDB
- `KMS_KEY_ID`: ID de votre clé KMS

Stratégie de clé KMS dans le compte DynamoDB

La politique AWS KMS clé suivante permet l'accès entre comptes entre vos services DynamoDB et Amazon Redshift. Dans cet exemple, le compte 444455556666 contient la table AWS KMS et la clé DynamoDB, tandis que le compte 111122223333 contient le cluster Amazon Redshift qui doit avoir accès pour déchiffrer les données.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

Politique IAM pour le rôle Amazon Redshift (dans le compte Amazon Redshift)

La politique IAM suivante permet à un service Amazon Redshift d'accéder aux tables DynamoDB et aux clés de chiffrement AWS KMS associées dans le cadre d'un scénario entre comptes. Dans cet exemple, le compte 444455556666 contient les ressources et les AWS KMS clés DynamoDB auxquelles le service Amazon Redshift doit accéder.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:us-east-1:444455556666:key/KMS_KEY_ID"
```

```
        }
    ]
}
```

Relation de confiance pour le rôle Amazon Redshift

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Politique de table DynamoDB (si vous utilisez des politiques basées sur les ressources)

La politique basée sur les ressources suivante permet à un service Amazon Redshift du compte 111122223333 d'accéder aux tables DynamoDB et aux flux du compte 444455556666. Joignez cette politique à votre table DynamoDB pour activer l'accès intercompte.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
```

```
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

Considérations importantes

1. Vérifiez que la clé KMS se situe dans la même région que votre table DynamoDB.
2. La clé KMS doit être une clé gérée par le client (CMK) et non une Clé gérée par AWS.
3. Si vous utilisez des tables globales DynamoDB, configurez les autorisations pour toutes les régions pertinentes.
4. Envisagez d'ajouter des instructions de condition pour restreindre l'accès en fonction des points de terminaison d'un VPC ou des plages d'adresses IP.
5. Pour une sécurité renforcée, pensez à utiliser une condition `aws:PrincipalOrgID` pour restreindre l'accès à votre organisation.
6. Surveillez l'utilisation des clés KMS grâce à CloudTrail des CloudWatch indicateurs.

Intégration zéro ETL de DynamoDB à Amazon Redshift

L'intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift permet des analytiques fluides des données DynamoDB, sans aucun codage. Cette fonctionnalité entièrement gérée réplique automatiquement les tables DynamoDB dans une base de données Amazon Redshift afin que les utilisateurs puissent exécuter des requêtes SQL et des analytiques sur leurs données DynamoDB

sans avoir à configurer de processus ETL complexes. L'intégration fonctionne en répliquant les données de la table DynamoDB vers la base de données Amazon Redshift.

Pour configurer l'intégration, il suffit de spécifier une table DynamoDB comme source et une base de données Amazon Redshift comme cible. Lors de l'activation, l'intégration exporte la table DynamoDB complète pour alimenter la base de données Amazon Redshift. La durée nécessaire à ce processus initial dépend de la taille de la table DynamoDB. L'intégration zéro ETL réplique ensuite de manière incrémentielle les mises à jour de DynamoDB vers Amazon Redshift toutes les 15 à 30 minutes à l'aide des exportations incrémentielles DynamoDB. Cela signifie que les données DynamoDB répliquées dans Amazon Redshift sont conservées automatiquement. up-to-date

Une fois la configuration terminée, les utilisateurs peuvent analyser les données DynamoDB dans Amazon Redshift à l'aide de clients et d'outils SQL standard, sans affecter les performances des tables DynamoDB. En éliminant la complexité ETL, cette intégration zéro ETL permet de débloquer rapidement et facilement des informations issues de DynamoDB grâce aux fonctionnalités d'analytique et de machine learning d'Amazon Redshift.

Rubriques

- Conditions préalables à la création d'une intégration zéro ETL DynamoDB à Amazon Redshift
- Limitations concernant l'utilisation des intégrations zéro ETL de DynamoDB avec Amazon Redshift
- Création d'une intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift
- Affichage des intégrations zéro ETL de DynamoDB à Amazon Redshift
- Suppression des intégrations zéro ETL de DynamoDB à Amazon Redshift

Conditions préalables à la création d'une intégration zéro ETL DynamoDB à Amazon Redshift

1. Vous devez avoir créé votre table DynamoDB source et votre cluster Amazon Redshift cible avant de créer une intégration. Ces informations sont traitées dans les sections Étape 1 : Configuration d'une table DynamoDB source et Étape 2 : Création d'un entrepôt de données Amazon Redshift.
2. Une intégration zéro ETL entre Amazon DynamoDB et Amazon Redshift nécessite que la restauration (PITR) soit activée dans votre table DynamoDB Point-in-time source.
3. Pour les politiques basées sur les ressources, si vous créez l'intégration où votre table DynamoDB et l'entrepôt de données Amazon Redshift se trouvent dans le même compte, vous pouvez utiliser l'option Fix it for me lors de l'étape de création de l'intégration pour appliquer automatiquement les politiques de ressources requises à DynamoDB et à Amazon Redshift.

Si vous créez une intégration dans laquelle votre table DynamoDB et l'entrepôt de données Amazon Redshift se trouvent dans des comptes AWS différents, vous devez appliquer la politique de ressources suivante à votre table DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        },
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-
name/export/*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

Vous devrez peut-être également configurer la politique de ressources de votre entrepôt de données Amazon Redshift. Pour plus d'informations, consultez Configuration de l'autorisation à l'aide de l'API Amazon Redshift.

4. Politiques basées sur l'identité :

   a. L'utilisateur qui crée l'intégration a besoin d'une politique basée sur l'identité qui autorise les actions suivantes : `GetResourcePolicy`, `PutResourcePolicy` et `UpdateContinuousBackups`.

   > **Note**
   >
   > Les exemples de politique suivants montreront la ressource sous la forme `arn:aws:redshift{-serverless}`. Cet exemple montre que l'arn peut être `arn:aws:redshift` ou `arn:aws:redshift-serverless` selon que votre espace de noms est un cluster Amazon Redshift ou un espace de noms Amazon Redshift sans serveur.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
```

```
        }
    ]
}
```

b. L'utilisateur chargé de configurer l'espace de noms Amazon Redshift de destination a besoin d'une politique basée sur l'identité qui autorise les actions suivantes : `PutResourcePolicy`, `DeleteResourcePolicy` et `GetResourcePolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        }
    ]
}
```

5. Autorisations pour le chiffrement des clés

Si la table DynamoDB source est chiffrée à l'aide d'une cléAWS KMS gérée par le client, vous devez ajouter la politique suivante à votre clé KMS. Cette politique permet à Amazon Redshift d'exporter les données de votre table chiffrée à l'aide de votre clé KMS.

```
{
    "Sid": "Statement to allow Amazon Redshift service to perform Decrypt operation
 on the source DynamoDB Table",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "redshift.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "<account>"
        },
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:redshift:<region>:<account>:integration:*"
        }
    }
}
```

Vous pouvez également suivre les étapes décrites dans Bien démarrer avec les intégrations zéro ETL dans le guide de gestion Amazon Redshift pour configurer les autorisations de l'espace de noms Amazon Redshift.

Limitations concernant l'utilisation des intégrations zéro ETL de DynamoDB avec Amazon Redshift

Les limitations générales suivantes s'appliquent à la version actuelle de cette intégration. Ces limitations peuvent changer dans les versions ultérieures.

> **Note**
>
> Outre les limitations ci-dessous, consultez également les considérations générales relatives à l'utilisation d'intégrations zéro ETL. Consultez Considérations relatives au moment d'utiliser les intégrations zéro ETL avec Amazon Redshift dans le Guide de l'utilisateur Amazon Redshift Management.

- La table DynamoDB et le cluster Amazon Redshift doivent se trouver dans la même région.
- La table DynamoDB source doit être chiffrée avec une clé appartenant à Amazon ou gérée par le client.AWS KMS Le chiffrement géré par Amazon n'est pas pris en charge pour la table DynamoDB source.

Création d'une intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift

Avant de créer une intégration zéro ETL, vous devez d'abord configurer votre table DynamoDB source, puis l'entrepôt de données Amazon Redshift cible.

Étape 1 : Configuration d'une table DynamoDB source

Pour créer une intégration zéro ETL avec Amazon Redshift, vous devez point-in-time activer la restauration (PITR) sur votre table. Si PITR n'est pas activée, la console peut résoudre ce problème pendant le processus de configuration de l'intégration. Pour plus de détails sur la façon d'activer le PITR, consultez la section Point-in-time restauration.

Étape 2 : Création d'un entrepôt de données Amazon Redshift

Si vous ne disposez pas d'un entrepôt de données Amazon Redshift, vous pouvez en créer un. Pour créer un groupe de travail Amazon Redshift sans serveur, voir Création d'un groupe de travail avec un espace de noms. Pour créer un cluster Amazon Redshift, consultez Création d'un cluster.

Le paramètre enable_case_sensitive_identifier doit être activé dans le groupe de travail ou le cluster Amazon Redshift cible pour que l'intégration aboutisse. Pour plus d'informations sur l'activation de la sensibilité à la casse, consultez Activation de la sensibilité à la casse pour votre entrepôt de données dans le Guide de la gestion Amazon Redshift.

Une fois la configuration du groupe de travail ou du cluster Amazon Redshift terminée, vous devez configurer votre entrepôt de données. Pour plus d'informations, consultez Zero-ETL integrations dans le Guide de gestion Amazon Redshift.

Étape 3 : Création d'une intégration zéro ETL DynamoDB

Avant de créer une intégration zéro ETL, veillez à effectuer les tâches décrites dans la section intitulée Conditions préalables à la création d'une intégration zéro ETL DynamoDB à Amazon Redshift. La création d'une intégration entre DynamoDB et Amazon Redshift est un processus en deux étapes. Créez d'abord une intégration à partir de DynamoDB, puis associez une base de données Amazon Redshift à cette nouvelle intégration.

Création d'une intégration zéro ETL

1. Connectez-vous à la console de AWS gestion et ouvrez la console Amazon DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodbv2
2. Dans le volet de navigation, choisissez Intégrations.
3. Sélectionnez Créer une intégration zéro ETL et choisissez Amazon Redshift.
4. Vous serez redirigé vers la console Amazon Redshift. Pour continuer la procédure, consultez la section DynamoDB dans Create a zero-ETL integration for DynamoDB.

Affichage des intégrations zéro ETL de DynamoDB à Amazon Redshift

Vous pouvez afficher les détails d'une intégration zéro ETL pour voir ses informations de configuration et son statut actuel.

Pour afficher les détails d'une intégration zéro ETL dans la console Amazon DynamoDB :

1. Connectez-vous à la console de AWS gestion et ouvrez la console Amazon DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodbv2
2. Dans la console DynamoDB, sélectionnez Intégrations.
3. Dans le volet Intégration zéro ETL, sélectionnez l'intégration zéro ETL que vous souhaitez consulter.

Pour afficher les détails d'une intégration zéro ETL dans la console Amazon Redshift :

1. Connectez-vous à la console de AWS gestion et ouvrez la console Amazon Redshift à l'adresse. https://console.aws.amazon.com/redshiftv2
2. Suivez les étapes décrites dans Affichage des intégrations zéro ETL.

> **Note**
>
> Les statuts possibles d'une intégration zéro ETL avec Amazon Redshift sont répertoriés dans la section Affichage des intégrations zéro ETL du Guide de gestion Amazon Redshift.

Suppression des intégrations zéro ETL de DynamoDB à Amazon Redshift

Lorsque vous supprimez une intégration zéro ETL, vos données ne sont pas supprimées de DynamoDB ou d'Amazon Redshift, mais DynamoDB arrête d'envoyer les données de votre table source à la cible Amazon Redshift.

Pour supprimer une intégration zéro ETL

1. Connectez-vous à la console de AWS gestion et ouvrez la console Amazon DynamoDB à l'adresse. https://console.aws.amazon.com/dynamodbv2
2. Dans la console DynamoDB, sélectionnez Intégrations.
3. Dans le volet Intégration zéro ETL, sélectionnez l'intégration zéro ETL que vous souhaitez supprimer.
4. Choisissez Gérer. Vous serez redirigé vers la page des détails de l'intégration.
5. Pour confirmer la suppression, choisissez Supprimer.

Chargement de données de DynamoDB dans Amazon Redshift à l'aide de la commande COPY

Amazon Redshift est compatible avec Amazon DynamoDB grâce à des capacités de business intelligence avancées et une interface SQL puissante. Lorsque vous copiez les données d'une table DynamoDB vers Amazon Redshift, vous pouvez exécuter des requêtes d'analyse de données complexes sur ces données, ainsi que des jointures avec d'autres tables de votre cluster Amazon Redshift.

En matière de débit provisionné, une opération de copie d'une table DynamoDB est comptabilisée par rapport à la capacité de lecture de la table. Une fois les données copiées, vos requêtes SQL dans Amazon Redshift n'affectent en rien DynamoDB. Cela est dû au fait que vos requêtes agissent sur une copie des données de DynamoDB, plutôt que sur DynamoDB lui-même.

Avant de pouvoir charger les données d'une table DynamoDB, vous devez créer une table Amazon Redshift pour faire office de destination pour les données. Gardez à l'esprit que vous copiez les données d'un environnement NoSQL vers un environnement SQL, et qu'il existe certaines règles dans un environnement qui ne s'appliquent pas dans l'autre. Voici quelques exemples de différences à prendre en compte :

- Les noms de table DynamoDB peuvent contenir jusqu'à 255 caractères, y compris « . » (point) et '-' (tiret), et sont sensibles à la casse. Les noms de table Amazon Redshift sont limités à 127 caractères, ne peuvent pas contenir de points ou de tirets, et ne sont pas sensibles à la casse. En outre, les noms de table ne peuvent pas entrer en conflit avec les mots réservés Amazon Redshift.
- DynamoDB ne prend pas en charge le concept SQL de NULL. Vous devez spécifier comment Amazon Redshift interprète les valeurs d'attribut vides ou vides dans DynamoDB, en les traitant comme des champs vides ou comme des champs vides. NULLs
- Les types de données DynamoDB ne correspondent pas directement à ceux d'Amazon Redshift. Vous devez vous assurer que chaque colonne de la table Amazon Redshift a le type de données et la taille appropriés pour contenir les données de DynamoDB.

Voici un exemple de commande COPY d'Amazon Redshift SQL :

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-
Access-Key>'
readratio 50;
```

Dans cet exemple, la table source dans DynamoDB est `my-favorite-movies-table`. La table cible dans Amazon Redshift est `favoritemovies`. La clause `readratio 50` règle le pourcentage de débit provisionné qui est consommé ; dans ce cas, la commande COPY n'utilise pas plus de 50 % des unités de capacité en lecture provisionnées pour `my-favorite-movies-table`. Nous vous recommandons hautement de définir ce ratio avec une valeur inférieure au débit moyen alloué non utilisé.

Pour obtenir des instructions détaillées sur le chargement de données de DynamoDB dans Amazon Redshift, consultez les sections suivantes dans le Manuel du développeur de bases de données Amazon Redshift :

- Chargement des données à partir d'une table DynamoDB
- Commande COPY
- Exemples COPY

Traitement de données DynamoDB avec Apache Hive sur Amazon EMR

Amazon DynamoDB est intégré avec Apache Hive, une application d'entreposage de données qui s'exécute sur Amazon EMR. Hive peut lire et écrire des données dans des tables DynamoDB, ce qui vous permet d'effectuer les opérations suivantes :

- Interroger des données DynamoDB en direct à l'aide d'un langage de type SQL (HiveQL).
- Copier des données d'une table DynamoDB vers un compartiment Amazon S3, et vice-versa.
- Copier les données d'une table DynamoDB dans le système de fichiers distribué Hadoop (HDFS) et inversement.
- Effectuer des opérations de jointure sur des tables DynamoDB.

Rubriques

- Présentation
- Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive
- Création d'une table externe dans Hive
- Traitement des instructions HiveQL
- Interrogation de données dans DynamoDB
- Copie de données depuis et vers Amazon DynamoDB
- Personnalisation de performances

Présentation

Amazon EMR est un service qui facilite le traitement rapide et économique de grandes quantités de données. Pour utiliser Amazon EMR, vous lancez un cluster géré d'instances Amazon EC2 exécutant l'infrastructure open source Hadoop. Hadoop est une application distribuée qui implémente l'algorithme MapReduce, où une tâche est mappée à plusieurs nœuds du cluster. Chaque nœud traite le travail qui lui est attribué, en parallèle avec d'autres nœuds. Enfin, les sorties sont réduites sur un seul nœud, ce qui donne le résultat final.

Vous pouvez choisir de lancer votre cluster Amazon EMR de façon à ce qu'il soit permanent ou temporaire :

- Un cluster permanent s'exécute jusqu'à ce que vous l'arrêtiez. Les clusters permanents sont idéaux pour l'analyse et l'entreposage de données, ainsi que pour toute autre utilisation interactive.
- Un cluster temporaire s'exécute le temps de traiter un flux de travail, puis s'arrête automatiquement. Les clusters temporaires sont idéaux pour des tâches de traitement périodiques, telles que l'exécution de scripts.

Pour plus d'informations sur l'architecture et l'administration d'Amazon EMR, consultez le Guide de gestion Amazon EMR.

Lorsque vous lancez un cluster Amazon EMR, spécifiez le nombre initial et le type d'instances Amazon EC2. Vous spécifiez également d'autres applications distribuées (en plus de Hadoop) que vous souhaitez exécuter sur le cluster. Ces applications sont Hue, Mahout, Pig, Spark etc.

Pour plus d'informations sur les applications pour Amazon EMR, consultez le Guide de version Amazon EMR.

Selon la configuration du cluster, vous pouvez disposer d'un ou plusieurs des types de nœuds suivants :

- Nœud leader – Gère le cluster en coordonnant la distribution de l'exécutable MapReduce et des sous-ensembles de données brutes vers les groupes d'instances de base et de tâche. Suit également le statut de chaque tâche exécutée et surveille l'intégrité des groupes d'instances. Un cluster ne contient qu'un seul nœud leader.
- Nœud de base – Exécute des tâches MapReduce et stocke les données à l'aide du système de fichiers distribué Hadoop (HDFS).
- Nœud de tâche (facultatif) – Exécute une tâche MapReduce.

Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive

Dans ce didacticiel, vous allez lancer un cluster Amazon EMR, puis utiliser Apache Hive pour traiter les données stockées dans une table DynamoDB.

Hive est une application d'entrepôt de données pour Hadoop, qui vous permet de traiter et d'analyser des données de plusieurs sources. Hive fournit un langage similaire à SQL, HiveQL, qui vous permet d'utiliser des données stockées localement dans le cluster Amazon EMR ou dans une source de données externe (telle qu'Amazon DynamoDB).

Pour plus d'informations, consultez le Didacticiel Hive.

Rubriques

- Avant de commencer
- Étape 1 : Créer une paire de clés Amazon EC2
- Étape 2 : lancer un cluster Amazon EMR
- Étape 3 : se connecter au nœud leader
- Étape 4 : charger des données dans HDFS
- Étape 5 : copier des données dans DynamoDB
- Étape 6 : interroger les données dans la table DynamoDB
- Étape 7 : (Facultatif) nettoyer

Avant de commencer

Pour ce didacticiel, vous avez besoin des éléments suivants :

- Un compte AWS. Si vous n'en avez pas, consultez S'inscrire àAWS.
- Un client SSH (Secure Shell). Vous utilisez le client SSH pour vous connecter au nœud leader du cluster Amazon EMR et exécuter des commandes interactives. Les clients SSH sont disponibles par défaut sur la plupart des installations Linux, Unix et Mac OS X. Les utilisateurs de Windows peuvent télécharger et installer le client PuTTY qui prend en charge SSH.

Étape suivante

Étape 1 : Créer une paire de clés Amazon EC2

Étape 1 : Créer une paire de clés Amazon EC2

Dans cette étape, vous allez créer la paire de clés Amazon EC2 dont vous avez besoin pour vous connecter à un nœud leader Amazon EMR et exécuter les commandes Hive.

1. Connectez-vous à la AWS Management Console et ouvrez la console Amazon EC2 à l'adresse https://console.aws.amazon.com/ec2/.
2. Choisissez une période (par exemple, `US West (Oregon)`). Il doit s'agir de la région dans laquelle se trouve votre table DynamoDB.
3. Dans le volet de navigation, cliquez sur Key Pairs.

4. Choisissez Create Key Pair (Créer une paire de clés).
5. Dans Key pair name (Nom de la paire de clés), tapez un nom pour votre paire de clés (par exemple, `mykeypair`), puis choisissez Create (Créer).
6. Téléchargez le fichier de clé privée. Le nom de fichier se terminera par `.pem` (par exemple, `mykeypair.pem`). Enregistrez ce fichier de clé privée en lieu sûr. Il est nécessaire pour accéder à tout cluster Amazon EMR lancé avec cette paire de clés.

   > **Important**
   >
   > Sans la paire de clés, vous ne pouvez pas vous connecter au nœud leader de votre cluster Amazon EMR.

   Pour plus d'informations sur les paires de clés, consultez Paires de clés Amazon EC2 dans le Guide de l'utilisateur Amazon EC2.

Étape suivante

Étape 2 : lancer un cluster Amazon EMR

Étape 2 : lancer un cluster Amazon EMR

Dans cette étape, vous allez configurer et lancer un cluster Amazon EMR. Hive et un gestionnaire de stockage pour DynamoDB sont déjà installés sur le cluster.

1. Ouvrez la console Amazon EMR à l'adresse https://console.aws.amazon.com/emr.
2. Choisissez Create Cluster (Créer un cluster).
3. Dans la page Create Cluster - Quick Options (Créer un cluster - Options rapides), procédez comme suit :

   a. Dans Cluster Name (Nom du cluster), saisissez un nom pour votre cluster (par exemple, `My EMR cluster`).
   b. Dans EC2 key pair (paire de clés EC2), choisissez la paire de clés que vous avez créée précédemment.

   Conservez les valeurs par défaut des autres paramètres.
4. Choisissez Create Cluster (Créer un cluster).

Le lancement de votre cluster prend plusieurs minutes. Vous pouvez utiliser la page Cluster Details (Détails du cluster) dans la console Amazon EMR pour surveiller sa progression.

Lorsque l'état du cluster est `Waiting`, cela signifie que la cluster est prêt pour utilisation.

Fichiers journaux de cluster et Amazon S3

Un cluster Amazon EMR génère des fichiers journaux contenant des informations sur l'état du cluster et des informations de débogage. Les paramètres par défaut pour Create Cluster - Quick Options (Créer un cluster - Options rapides) incluent la configuration de la journalisation Amazon EMR.

À défaut de compartiment Amazon S3, la AWS Management Console en crée un. Le nom du compartiment est `aws-logs-`*`account-id`*`-`*`region`*, où *`account-id`* est votre numéro de compte AWS et *`region`* la région dans laquelle vous avez lancé le cluster (par exemple, `aws-logs-123456789012-us-west-2`).

> **Note**
>
> Vous pouvez utiliser la console Amazon S3 pour afficher les fichiers journaux. Pour plus d'informations, consultez Afficher les fichiers journaux dans le Guide de gestion Amazon EMR.

Vous pouvez utiliser ce compartiment à des fins autres que la journalisation. Par exemple, vous pouvez utiliser le compartiment comme emplacement pour stocker un script Hive ou comme destination lors de l'exportation de données d'Amazon DynamoDB vers Amazon S3.

Étape suivante

Étape 3 : se connecter au nœud leader

Étape 3 : se connecter au nœud leader

Lorsque l'état de votre cluster Amazon EMR est `Waiting`, vous pouvez vous connecter au nœud leader en utilisant le protocole SSH, et effectuer des opérations de ligne de commande.

1. Dans la console Amazon EMR, choisissez le nom de votre cluster pour afficher son état.
2. Dans la page Cluster Details (Détails du cluster), recherchez le champ Leader public DNS (DNS public leader). Il s'agit du nom DNS public du nœud leader de votre cluster Amazon EMR.

3. À droite du nom DNS, choisissez le lien SSH.
4. Suivez les instructions fournies dans Connexion au nœud leader à l'aide de SSH.

   En fonction de votre système d'exploitation, choisissez l'onglet Windows ou Mac/Linux, puis suivez les instructions de connexion au nœud leader.

Une fois connecté au nœud leader à l'aide de SSH ou de PuTTY, vous devriez voir une invite de commande similaire à la suivante :

```
[hadoop@ip-192-0-2-0 ~]$
```

Étape suivante

Étape 4 : charger des données dans HDFS

Étape 4 : charger des données dans HDFS

Dans cette étape, vous allez copier un fichier de données dans Hadoop Distributed File System (HDFS), puis créer une table Hive externe qui mappe à ce fichier de données.

Télécharger l'échantillon de données

1. Téléchargez l'archive de l'échantillon de données (`features.zip`) :

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Extrayez le fichier `features.txt` de l'archive :

   ```
   unzip features.zip
   ```

3. Affichez les premières lignes du fichier `features.txt` :

   ```
   head features.txt
   ```

   Le résultat doit ressembler à ceci :

   ```
   1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
   875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
   ```

```
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

Le fichier `features.txt` contient un sous-ensemble de données du United States Board on Geographic Names (http://geonames.usgs.gov/domestic/download_data.htm). Les champs de chaque ligne représentent les éléments suivants :

- ID de fonction (identifiant unique)
- Nom
- Classe (lac, forêt, rivière, etc.)
- État
- Latitude (degrés)
- Longitude (degrés)
- Altitude (pieds)

4. A partir d'une invite de commande, entrez la commande suivante :

```
hive
```

L'invite de commande devient : `hive>`.

5. Entrez l'instruction HiveQL suivante pour créer une table Hive native :

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. Entrez l'instruction HiveQL suivante pour charger la table avec les données :

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Vous avez maintenant une table Hive native remplie des données du fichier `features.txt`. Pour vérifier, entrez l'instruction HiveQL suivante :

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

   La sortie doit afficher une liste d'États et le nombre d'entités géographiques dans chacun d'eux.

Étape suivante

Étape 5 : copier des données dans DynamoDB

Étape 5 : copier des données dans DynamoDB

Dans cette étape, vous allez copier les données de la table Hive (`hive_features`) vers une nouvelle table dans DynamoDB.

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Choisissez Créer une table.
3. Dans l'écran Create DynamoDB table (Créer une table DynamoDB), procédez comme suit :
   a. Dans Table, saisissez **`Features`**.
   b. Pour Primary key (Clé primaire), dans le champ Partition key (Clé de partition), saisissez **`Id`**. Définissez le type de données sur Number (Nombre).

      Effacez le contenu du champ Use Default Settings (Utiliser les paramètres par défaut). Pour Provisioned Capacity (Capacité provisionnée), saisissez ce qui suit :

      - Read Capacity Units (Unités de capacité de lecture) – `10`
      - Write Capacity Units (Unités de capacité d'écriture) – `10`

Sélectionnez Create (Créer).

4. À l'invite Hive, saisissez l'instruction HiveQL suivante :

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
    feature_name  STRING,
    feature_class STRING,
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

Vous avez maintenant établi un mappage entre Hive et la table Features (Particularités) dans DynamoDB.

5. Entrez l'instruction HiveQL suivante pour importer des données dans DynamoDB :

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

Hive soumet une tâche MapReduce qui sera traitée par votre cluster Amazon EMR. L'accomplissement de la tâche peut prendre plusieurs minutes.

6. Vérifiez que les données ont été chargées dans DynamoDB :

   a. Dans le panneau de navigation de la console DynamoDB, choisissez Tables.

b. Choisissez le tableau Fonctions, puis l'onglet Items (Eléments) pour afficher les données.

Étape suivante

Étape 6 : interroger les données dans la table DynamoDB

Étape 6 : interroger les données dans la table DynamoDB

Dans cette étape, vous allez utiliser HiveQL pour interroger la table Features (Particularités) dans DynamoDB. Essayez les requêtes Hive suivantes :

1. Tous les types de particularités (`feature_class`) par ordre alphabétique :

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Tous les lacs dont le nom commence par la lettre « M » :

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. États dont au moins trois particularités sont à une altitude supérieure à un mile (5 280 pieds) :

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

Étape suivante

Étape 7 : (Facultatif) nettoyer

Étape 7 : (Facultatif) nettoyer

Maintenant que vous avez terminé le didacticiel, vous pouvez continuer à lire cette section pour en savoir plus sur l'utilisation des données DynamoDB dans Amazon EMR. Pendant ce temps, vous pouvez garder votre cluster Amazon EMR opérationnel.

Si vous n'avez plus besoin du cluster, résiliez-le et supprimez les ressources associées. Vous éviterez ainsi d'être facturé pour des ressources dont vous n'avez pas besoin.

1. Résiliez le cluster Amazon EMR :
   a. Ouvrez la console Amazon EMR à l'adresse https://console.aws.amazon.com/emr.
   b. Choisissez le cluster Amazon EMR, Terminate (Résilier), puis confirmez.
2. Supprimez la table Features dans DynamoDB :
   a. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
   b. Dans le volet de navigation, choisissez Tables.
   c. Choisissez la table Features. Dans le menu Actions, choisissez Delete Table (Supprimer la table).
3. Supprimez le compartiment Amazon S3 contenant les fichiers journaux Amazon EMR :
   a. Ouvrez la console Amazon S3 sur https://console.aws.amazon.com/s3/.
   b. Dans la liste de compartiments, choisissez `aws-logs-` *`accountID`*`-`*`region`*, où *`accountID`* est le numéro de votre compte AWS, et*`region`* la région dans laquelle vous avez lancé le cluster.
   c. Dans le menu Action, choisissez Delete (Supprimer).

Création d'une table externe dans Hive

Dans Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive, vous avez créé une table Hive externe mappée à une table DynamoDB. Lorsque vous avez émis des instructions HiveQL par rapport à la table externe, les opérations de lecture et d'écriture ont été transmises à la table DynamoDB.

Vous pouvez considérer une table externe comme un pointeur vers une source de données gérée et stockée ailleurs. Dans ce cas, la source de données sous-jacente est une table DynamoDB (la table doit être pré-existante car vous ne pouvez pas créer, mettre à jour ou supprimer une table

DynamoDB à partir de Hive). Vous utilisez l'instruction `CREATE EXTERNAL TABLE` pour créer la table externe. Ensuite, vous pouvez utiliser HiveQL pour utiliser les données dans DynamoDB comme si elles étaient stockées localement dans Hive.

> **Note**
>
> Vous pouvez utiliser des instructions `INSERT` pour insérer des données dans une table externe, et des instructions `SELECT` pour sélectionner des données dans celle-ci. Toutefois, vous ne pouvez pas utiliser des instructions `UPDATE` ou `DELETE` pour manipuler des données dans la table.

Si vous n'avez plus besoin de la table externe, vous pouvez la supprimer à l'aide de l'instruction `DROP TABLE`. Dans ce cas, l'instruction `DROP TABLE` supprime uniquement la table externe dans Hive. Elle n'affecte ni la table DynamoDB sous-jacente ni aucune de ses données.

Rubriques

- Syntaxe de CREATE EXTERNAL TABLE
- Mappage de types de données

Syntaxe de CREATE EXTERNAL TABLE

Cette section présente la syntaxe HiveQL pour créer une table Hive externe qui mappe à une table DynamoDB :

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

La ligne 1 est le début de l'instruction `CREATE EXTERNAL TABLE` où vous indiquez le nom de la table Hive (hive_table) que vous souhaitez créer.

La ligne 2 spécifie les colonnes et types de données pour hive_table. Vous devez définir des colonnes et des types de données correspondant aux attributs de la table DynamoDB.

La ligne 3 est la clause `STORED BY` dans laquelle vous spécifiez une classe qui assure la gestion des données entre Hive et la table DynamoDB. Pour DynamoDB, `STORED BY` doit être défini sur `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`.

La ligne 4 est le début de la clause `TBLPROPERTIES` où vous définissez les paramètres suivants pour `DynamoDBStorageHandler` :

- `dynamodb.table.name` – Le nom de la table DynamoDB.
- `dynamodb.column.mapping` – Paires de noms de colonnes dans la table Hive et leurs attributs correspondants dans la table DynamoDB. Chaque paire présente la forme nom_colonne_hive:nom_attribut_dynamodb, et les paires sont séparées par des virgules.

Remarques :

- Le nom du nom de la table Hive ne doit pas nécessairement être identique au nom de la table DynamoDB.
- Les noms de colonnes de la table Hive ne doivent pas nécessairement être identiques à ceux de la table DynamoDB.
- La table spécifiée par `dynamodb.table.name` doit exister dans DynamoDB.
- Dans `dynamodb.column.mapping` :
  - Vous devez mapper les attributs de schéma de clé pour la table DynamoDB. Cela inclut la clé de partition et la clé de tri (si elle est présente).
  - Vous n'avez pas besoin de mapper les attributs autres que de clé de la table DynamoDB. Toutefois, l'interrogation de la table Hive n'affiche pas les données de ces attributs .
  - Si les types de données d'une colonne de la table Hive et d'un attribut DynamoDB sont incompatibles, `NULL` s'affiche dans ces colonnes lorsque vous interrogez la table Hive.

> **Note**
>
> L'instruction `CREATE EXTERNAL TABLE` n'effectue aucune validation sur la clause `TBLPROPERTIES`. Les valeurs que vous fournissez pour `dynamodb.table.name`

et `dynamodb.column.mapping` ne sont évaluées que par la classe `DynamoDBStorageHandler` lorsque vous tentez d'accéder à la table.

Mappage de types de données

Le tableau suivant présente les types de données DynamoDB et les types de données Hive compatibles :

Type de données DynamoDB	Type de données Hive
Chaîne	`STRING`
Nombre	`BIGINT` ou `DOUBLE`
Binaire	`BINARY`
Ensemble de chaînes	`ARRAY<STRING>`
Ensemble de nombres	`ARRAY<BIGINT>` ou `ARRAY<DOUBLE>`
Ensemble de binaires	`ARRAY<BINARY>`

Note

La classe `DynamoDBStorageHandler` ne prend pas en charge les types de données DynamoDB suivants, qui ne peuvent donc pas être utilisés avec `dynamodb.column.mapping` :

- Map
- Liste
- Booléen
- Null

Toutefois, si vous devez travailler avec ces types de données, vous pouvez créer une entité unique nommée `item` qui représente l'ensemble de l'élément DynamoDB sous la forme

d'une carte de chaînes pour les clés et les valeurs de la carte. Pour plus d'informations, consultez Copie de données sans mappage de colonnes.

Si vous souhaitez mapper un attribut DynamoDB de type Number, vous devez choisir un type Hive approprié :

- Le type Hive `BIGINT` est destiné aux entiers signés de 8 octets. Il est identique au type de données `long` en Java.
- Le type Hive `DOUBLE` est destiné aux nombres à virgule flottante double précision de 8 bits. Il est identique au type `double` en Java.

Si vous avez des données numériques stockées dans DynamoDB dont la précision est supérieure à celle du type de données Hive que vous choisissez, l'accès aux données DynamoDB risque d'entraîner une perte de précision.

Si vous exportez des données de type Binary de DynamoDB vers (Amazon S3) ou HDFS, les données sont stockées sous forme de chaîne codée en base 64. Si vous importez des données d'Amazon S3 ou de HDFS dans le type Binary DynamoDB, vous devez vous assurer que les données sont codées sous forme de chaîne en base 64.

Traitement des instructions HiveQL

Hive est une application s'exécutant sur Hadoop, qui est une infrastructure orientée lots pour exécuter des tâches MapReduce. Lorsque vous émettez une instruction HiveQL, Hive détermine s'il peut renvoyer les résultats immédiatement ou s'il doit soumettre une tâche MapReduce.

Prenons l'exemple de la table ddb_features (extraite de Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive). La requête Hive suivante affiche les abréviations d'État et le nombre de sommets dans chacun d'eux :

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive ne retourne pas les résultats immédiatement. Au lieu de cela, il soumet une tâche MapReduce est traitée par l'infrastructure Hadoop. Hive attend que le travail soit terminé avant d'afficher les résultats de la requête :

```
AK  2
AL  2
AR  2
AZ  3
CA  7
CO  2
CT  2
ID  1
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

Surveillance et annulation de tâches

Lorsque Hive lance une tâche Hadoop, il affiche une sortie de cette tâche. L'état d'accomplissement de la tâche est mis à jour à mesure que la tâche progresse. Dans certains cas, il se peut que l'état ne soit pas mis à jour pendant une longue période (cela peut se produire lorsque vous interrogez une table DynamoDB de grande taille dont le paramètre de capacité de lecture provisionnée est faible).

Si vous devez annuler la tâche avant la fin de son exécution, vous pouvez taper **`Ctrl+C`** à tout moment.

Interrogation de données dans DynamoDB

Les exemples suivants présentent différentes façons d'utiliser HiveQL pour interroger des données stockées dans DynamoDB.

Ces exemples font référence à la table ddb_features dans le didacticiel (Étape 5 : copier des données dans DynamoDB).

Rubriques

- Utilisation de fonctions d'agrégation
- Utilisation des clauses GROUP BY et HAVING
- Jointure de deux tables DynamoDB
- Jointure de tables de sources différentes

Utilisation de fonctions d'agrégation

HiveQL fournit des fonctions intégrées pour la synthèse des valeurs de données. Vous pouvez par exemple utiliser la fonction `MAX` pour rechercher la plus grande valeur dans une colonne sélectionnée. L'exemple suivant renvoie l'altitude de la particularité la plus élevée dans l'État du Colorado.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Utilisation des clauses GROUP BY et HAVING

Vous pouvez utiliser la clause `GROUP BY` pour collecter les données de plusieurs enregistrements. Elle est souvent utilisé avec une fonction d'agrégation telle que `SUM`, `COUNT`, `MIN` ou `MAX`. Vous pouvez également utiliser la clause `HAVING` pour écarter tous les résultats ne répondant pas à certains critères.

L'exemple suivant renvoie la liste des altitudes les plus élevées des États qui comptent plus de cinq particularités dans la table ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Jointure de deux tables DynamoDB

L'exemple suivant mappe une autre table Hive (east_coast_states) à une table dans DynamoDB. L'instruction SELECT opère une jointure entre ces deux tables. La jointure est calculée sur le cluster, puis renvoyée. La jointure n'a pas lieu dans DynamoDB.

Considérons une table DynamoDB nommée EastCoastStates contenant les données suivantes :

```
StateName       StateAbbrev

Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
Florida         FL
```

Supposons que la table est disponible en tant que table externe Hive nommée east_coast_states :

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

La jointure suivante renvoie les États de la côte Est des États-Unis qui possèdent au moins trois particularités :

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

Jointure de tables de sources différentes

Dans l'exemple suivant, s3_east_coast_states est une table Hive associée à un fichier CSV stocké dans Amazon S3. La table ddb_features est associée à des données dans DynamoDB. L'exemple suivant joint ces deux tables pour renvoyer les particularités géographiques des États dont le nom commence par « New ».

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

Copie de données depuis et vers Amazon DynamoDB

Dans Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive, vous avez copié des données d'une table Hive native dans une table DynamoDB externe, puis interrogé la table DynamoDB externe. La table est externe car elle existe en dehors de Hive. Même si vous supprimez la table Hive source du mappage, la table dans DynamoDB n'est pas affectée.

Hive est une excellente solution pour copier des données entre des tables DynamoDB, des compartiments Amazon S3, des tables Hive natives et un système de fichiers distribué Hadoop (HDFS). Cette section fournit des exemples de ces opérations.

Rubriques

- Copie de données entre DynamoDB et une table Hive native
- Copie de données entre DynamoDB et Amazon S3
- Copie de données entre DynamoDB et HDFS
- En utilisant la compression des données
- Lecture de données en caractères UTF-8 non affichables

Copie de données entre DynamoDB et une table Hive native

Si vous avez des données dans une table DynamoDB, vous pouvez copier les données vers une table Hive native. Cela vous donne un instantané des données au moment où vous les avez copiées.

Vous pourrez décider de procéder de la sorte si vous devez exécuter de nombreuses requêtes HiveQL, mais ne souhaitez pas utiliser la capacité de débit provisionnée de DynamoDB. Étant donné que les données de la table Hive native sont une copie des données de DynamoDB, et non des données dynamiques, vos requêtes ne doivent pas s'attendre à ce que les données soient à jour.

> **Note**
>
> Les exemples fournis dans cette section partent du principe que vous avez suivi les étapes décrites dans Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive et disposez d'une table externe dans DynamoDB nommée ddb_features.

Example De DynamoDB vers une table Hive native

Vous pouvez créer une table Hive native et la remplir des données de la table ddb_features comme suit :

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

Vous pouvez ensuite actualiser les données à tout moment :

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

Dans ces exemples, la sous-requête `SELECT * FROM ddb_features` extrait toutes les données de la table ddb_features. Si vous ne souhaitez copier qu'un sous-ensemble des données, vous pouvez utiliser une clause `WHERE` dans la sous-requête.

L'exemple suivant crée une table Hive native contenant uniquement certains des attributs pour les lacs et les sommets :

```
CREATE TABLE lakes_and_summits AS
```

```
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example D'une table Hive native à DynamoDB

Utilisez l'instruction HiveQL suivante pour copier les données de la table Hive native vers la table ddb_features :

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Copie de données entre DynamoDB et Amazon S3

Si vous disposez de données dans une table DynamoDB, vous pouvez utiliser Hive pour les copier vers un compartiment Amazon S3.

Vous pouvez le faire si vous souhaitez créer une archive de données dans votre table DynamoDB. Par exemple, supposons que vous disposez d'un environnement de test dans lequel vous devez utiliser un jeu de base de données de test dans DynamoDB. Vous pouvez copier les données de base dans un compartiment Amazon S3, puis exécuter vos tests. Ensuite, vous pouvez réinitialiser l'environnement de test en restaurant les données de base à partir du compartiment Amazon S3 vers DynamoDB.

Si vous avez travaillé dans Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive, vous disposez déjà d'un compartiment Amazon S3 contenant vos journaux Amazon EMR. Vous pouvez utiliser ce compartiment pour les exemples de cette section si vous connaissez le chemin d'accès racine du compartiment :

1. Ouvrez la console Amazon EMR à l'adresse https://console.aws.amazon.com/emr.
2. Pour Name (Nom), choisissez votre cluster.
3. L'URI figure dans Log URI (URI de journal) sous Configuration Details (Détails de configuration).
4. Notez le chemin d'accès racine du compartiment. La convention de dénomination est la suivante :

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   où *`accountID`* est l'ID de votre compte AWS et region la région AWS de votre compartiment.

> **Note**
>
> Pour ces exemples, nous allons utiliser un sous-chemin dans le compartiment, comme dans cet exemple :
> `s3://aws-logs-123456789012-us-west-2/hive-test`

Les procédures suivantes partent du principe que vous avez suivi les étapes décrites dans le didacticiel et disposez d'une table externe dans DynamoDB nommée ddb_features.

Rubriques

- Copie de données à l'aide du format par défaut de Hive
- Copie de données avec un format spécifié par l'utilisateur
- Copie de données sans mappage de colonnes
- Affichage des données dans Amazon S3

Copie de données à l'aide du format par défaut de Hive

Example De DynamoDB vers Amazon S3

Utilisez une instruction `INSERT OVERWRITE` pour écrire directement sur Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Le fichier de données dans Amazon S3 ressemble à ceci :

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Les champs sont séparés par un caractère SOH (début d'en-tête, 0x01). Dans le fichier, SOH s'affiche sous la forme **^A**.

Example D'Amazon S3 vers DynamoDB

1. Créez une table externe pointant vers les données non mises en forme dans Amazon S3.

```
CREATE EXTERNAL TABLE s3_features_unformatted
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. Copiez les données vers DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

Copie de données avec un format spécifié par l'utilisateur

Si vous souhaitez spécifier votre propre caractère séparateur de champs, vous pouvez créer une table externe qui mappe au compartiment Amazon S3. Vous pouvez utiliser cette technique pour créer des fichiers de données avec des valeurs séparées par des virgules (CSV).

Example De DynamoDB vers Amazon S3

1. Créez une table externe Hive qui mappe à Amazon S3. Lorsque vous effectuez cette opération, assurez-vous que les types de données sont cohérents avec ceux de la table externe DynamoDB.

```
CREATE EXTERNAL TABLE s3_features_csv
    (feature_id       BIGINT,
    feature_name      STRING,
    feature_class     STRING,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE,
    prim_long_dec     DOUBLE,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. Copiez les données à partir de DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Le fichier de données dans Amazon S3 ressemble à ceci :

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example D'Amazon S3 vers DynamoDB

Avec une seule instruction HiveQL, vous pouvez remplir la table DynamoDB à l'aide des données d'Amazon S3 :

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Copie de données sans mappage de colonnes

Vous pouvez copier des données de DynamoDB dans un format brut et les écrire dans Amazon S3 sans spécifier de types de données ou de mappage de colonnes. Vous pouvez utiliser cette méthode pour créer une archive des données DynamoDB et la stocker dans Amazon S3.

Example De DynamoDB vers Amazon S3

1. Créez une table externe associée à votre table DynamoDB. (Il n'y a pas de `dynamodb.column.mapping` dans cette instruction HiveQL.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Créez une autre table externe associée à votre compartiment Amazon S3.

   ```
   CREATE EXTERNAL TABLE s3_features_no_mapping
   ```

```
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

3. Copiez les données de DynamoDB vers Amazon S3.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Le fichier de données dans Amazon S3 ressemble à ceci :

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Chaque champ commence par un caractère STX (début de texte, 0x02) et se termine par un caractère ETX (fin de texte, 0x03). Dans le fichier, STX apparaît sous la forme **^B**, et ETX sous la forme **^C**.

Example D’Amazon S3 vers DynamoDB

Avec une seule instruction HiveQL, vous pouvez remplir la table DynamoDB à l’aide des données d’Amazon S3 :

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Affichage des données dans Amazon S3

Si vous utilisez SSH pour vous connecter au nœud leader, vous pouvez utiliser AWS Command Line Interface (AWS CLI) pour accéder aux données que Hive a écrites sur Amazon S3.

Les étapes suivantes partent du principe que vous avez copié les données de DynamoDB vers Amazon S3 en suivant l'une des procédures décrites dans cette section.

1. Si vous vous trouvez à l'invite de commande Hive, passez l'invite de commande Linux.

   ```
   hive> exit;
   ```

2. Répertoriez le contenu du répertoire hive-test dans votre compartiment Amazon S3. (C'est là que Hive a copié les données de DynamoDB.)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   La réponse devrait ressembler à ceci :

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   Le nom du fichier (000000_0) est généré par le système.

3. (Facultatif) Vous pouvez copier le fichier de données d'Amazon S3 vers le système de fichiers local sur le nœud principal. Après cela, vous pouvez vous servir d'utilitaires de ligne de commande Linux standard pour travailler avec les données dans le fichier.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   La réponse devrait ressembler à ceci :

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > **Note**
   >
   > Le système de fichiers local sur le nœud leader a une capacité limitée. N'utilisez pas cette commande avec des fichiers plus grands que l'espace disponible dans le système de fichiers local.

Copie de données entre DynamoDB et HDFS

Si vous disposez de données dans une table DynamoDB, vous pouvez utiliser Hive pour copier les données vers le système de fichiers distribué Hadoop (HDFS).

Vous pouvez faire cela si vous exécutez une tâche MapReduce nécessitant des données de DynamoDB. Si vous copiez les données de DynamoDB vers HDFS, Hadoop peut les traiter en utilisant tous les nœuds disponibles dans le cluster Amazon EMR en parallèle. Une fois la tâche MapReduce terminée, vous pouvez écrire les résultats de HDFS vers DDB.

Dans les exemples suivants, Hive lit et écrit dans le répertoire HDFS suivant : `/user/hadoop/hive-test`

> **Note**
>
> Les exemples fournis dans cette section partent du principe que vous avez suivi les étapes décrites dans Didacticiel : Utilisation d'Amazon DynamoDB et d'Apache Hive et disposez d'une table externe dans DynamoDB nommée ddb_features.

Rubriques

- Copie de données à l'aide du format par défaut de Hive
- Copie de données avec un format spécifié par l'utilisateur
- Copie de données sans mappage de colonnes
- Accès aux données dans HDFS

Copie de données à l'aide du format par défaut de Hive

Example De DynamoDB vers HDFS

Utilisez une instruction `INSERT OVERWRITE` pour écrire directement sur HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

Le fichier de données dans HDFS ressemble à ceci :

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Les champs sont séparés par un caractère SOH (début d'en-tête, 0x01). Dans le fichier, SOH s'affiche sous la forme **^A**.

Example De HDFS vers DynamoDB

1. Créez une table externe qui mappe à des données non mises en forme dans HDFS.

```
CREATE EXTERNAL TABLE hdfs_features_unformatted
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Copiez les données vers DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Copie de données avec un format spécifié par l'utilisateur

Si vous souhaitez utiliser un autre caractère séparateur de champs, vous pouvez créer une table externe qui mappe au répertoire HDFS. Vous pouvez utiliser cette technique pour créer des fichiers de données avec des valeurs séparées par des virgules (CSV).

Example De DynamoDB vers HDFS

1. Créez une table externe Hive qui mappe à HDFS. Lorsque vous effectuez cette opération, assurez-vous que les types de données sont cohérents avec ceux de la table externe DynamoDB.

```
CREATE EXTERNAL TABLE hdfs_features_csv
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
```

```
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Copiez les données à partir de DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

Le fichier de données dans HDFS ressemble à ceci :

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example De HDFS vers DynamoDB

Avec une seule instruction HiveQL, vous pouvez remplir la table DynamoDB à l'aide des données de HDFS :

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Copie de données sans mappage de colonnes

Vous pouvez copier des données de DynamoDB dans un format brut et les écrire dans HDFS sans spécifier de types de données ou de mappage de colonnes. Vous pouvez utiliser cette méthode pour créer une archive des données DynamoDB et la stocker dans HDFS.

> **Note**
>
> Si votre table DynamoDB contient des attributs de type Map, List, Boolean ou Null, c'est la seule façon d'utiliser Hive pour copier des données de DynamoDB vers HDFS.

Example De DynamoDB vers HDFS

1. Créez une table externe associée à votre table DynamoDB. (Il n'y a pas de `dynamodb.column.mapping` dans cette instruction HiveQL.)

```
CREATE EXTERNAL TABLE ddb_features_no_mapping
    (item MAP<STRING, STRING>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "Features");
```

2. Créez une autre table externe associée à votre répertoire HDFS.

```
CREATE EXTERNAL TABLE hdfs_features_no_mapping
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 'hdfs:///user/hadoop/hive-test';
```

3. Copiez les données de DynamoDB vers HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Le fichier de données dans HDFS ressemble à ceci :

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Chaque champ commence par un caractère STX (début de texte, 0x02) et se termine par un caractère ETX (fin de texte, 0x03). Dans le fichier, STX apparaît sous la forme **^B**, et ETX sous la forme **^C**.

Example De HDFS vers DynamoDB

Avec une seule instruction HiveQL, vous pouvez remplir la table DynamoDB à l'aide des données de HDFS :

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Accès aux données dans HDFS

HDFS est un système de fichiers distribué, accessible à tous les nœuds dans le cluster Amazon EMR. Si vous utilisez SSH pour vous connecter au nœud leader, vous pouvez utiliser des outils de ligne de commande pour accéder aux données que Hive a écrites sur HDFS.

HDFS n'est pas la même chose que le système de fichiers local sur le nœud leader. Vous ne pouvez pas utiliser des fichiers et répertoires dans HDFS à l'aide de commandes Linux standard (telles que `cat`, `cp`, `mv` ou `rm`). Au lieu de cela, vous effectuez ces tâches à l'aide de la commande `hadoop fs`.

Les étapes suivantes partent du principe que vous avez copié les données de DynamoDB vers HDFS en suivant l'une des procédures décrites dans cette section.

1. Si vous vous trouvez à l'invite de commande Hive, passez l'invite de commande Linux.

   ```
   hive> exit;
   ```

2. Répertoriez le contenu du répertoire /user/hadoop/hive-test dans HDFS. (C'est là que Hive a copié les données de DynamoDB.)

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   La réponse devrait ressembler à ceci :

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   Le nom du fichier (000000_0) est généré par le système.

3. Affichez le contenu du fichier :

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

> **Note**
>
> Dans cet exemple, le fichier est relativement petit (environ 29 Ko). Soyez prudent lorsque vous utilisez cette commande avec des fichiers très volumineux ou qui contiennent des caractères non affichables.

4. (Facultatif) Vous pouvez copier le fichier de données de HDFS vers le système de fichiers local sur le nœud leader. Après cela, vous pouvez vous servir d'utilitaires de ligne de commande Linux standard pour travailler avec les données dans le fichier.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Cette commande n'a pas pour effet d'écraser le fichier.

> **Note**
>
> Le système de fichiers local sur le nœud leader a une capacité limitée. N'utilisez pas cette commande avec des fichiers plus grands que l'espace disponible dans le système de fichiers local.

En utilisant la compression des données

Lorsque vous utilisez Hive pour copier des données entre différentes sources, vous pouvez demander une compression de données à la volée. Hive fournit plusieurs codecs de compression. Vous pouvez en choisir un au cours de votre session Hive. Lorsque procédez de la sorte, les données sont compressées au format spécifié.

L'exemple suivant compresse les données à l'aide de l'algorithme Lempel-Ziv-Oberhumer (LZO).

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
```

```
FROM hiveTableName;
```

Le fichier obtenu dans Amazon S3 a un nom généré par le système doté du suffixe `.lzo` (par exemple, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Les codecs de compression disponibles sont les suivants :

- `org.apache.hadoop.io.compress.GzipCodec`
- `org.apache.hadoop.io.compress.DefaultCodec`
- `com.hadoop.compression.lzo.LzoCodec`
- `com.hadoop.compression.lzo.LzopCodec`
- `org.apache.hadoop.io.compress.BZip2Codec`
- `org.apache.hadoop.io.compress.SnappyCodec`

Lecture de données en caractères UTF-8 non affichables

Pour lire et écrire des données en caractères UTF-8 non affichables, vous pouvez utiliser la clause `STORED AS SEQUENCEFILE` lorsque vous créez une table Hive. SequenceFile est un format de fichier binaire Hadoop. Vous devez utiliser Hadoop pour lire ce fichier. L'exemple suivant montre comment exporter des données de DynamoDB vers Amazon S3. Vous pouvez utiliser cette fonctionnalité pour gérer les caractères codés UTF-8 non affichables.

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

Personnalisation de performances

Lorsque vous créez une table externe Hive qui mappe à une table DynamoDB, vous ne consommez aucune capacité de lecture ou d'écriture de DynamoDB. Cependant, l'activité de lecture et d'écriture sur la table Hive (telle que `INSERT` ou `SELECT`) se traduit directement en opérations de lecture et d'écriture sur la table DynamoDB sous-jacente.

Apache Hive sur Amazon EMR implémente sa propre logique pour l'équilibrage de la charge d'E/S sur la table DynamoDB, et cherche à minimiser la possibilité de dépassement du débit provisionné

de la table. À la fin de chaque requête Hive, Amazon EMR renvoie les métriques d'exécution, dont le nombre de fois que votre débit approvisionné a été dépassé. Vous pouvez utiliser ces informations, ainsi que les métriques CloudWatch sur votre table DynamoDB, pour améliorer les performances des demandes suivantes.

La console Amazon EMR fournit des outils de surveillance de base pour votre cluster. Pour plus d'informations, consultez Affichage et surveillance d'un cluster dans le Guide de gestion Amazon EMR.

Vous pouvez également surveiller vos tâches de cluster et Hadoop à l'aide d'outils web tels que Hue, Ganglia et l'interface web Hadoop. Pour plus d'informations, consultez Affichage des interfaces web hébergées sur les clusters Amazon EMR dans le Guide de gestion Amazon EMR.

Cette section décrit la procédure à suivre pour ajuster les performances des opérations Hive sur des tables DynamoDB externes.

Rubriques

- Débit alloué DynamoDB
- Ajustement des mappeurs
- Rubriques supplémentaires

Débit alloué DynamoDB

Lorsque vous émettez des instructions HiveQL par rapport à la table DynamoDB externe, la classe `DynamoDBStorageHandler` effectue les demandes d'API DynamoDB de bas niveau appropriées qui consomment du débit approvisionné. Si la capacité de lecture ou d'écriture sur la table DynamoDB est insuffisante, la demande est limitée, ce qui ralentit les performances de HiveQL. Pour cette raison, vous devez vous assurer que la table dispose d'une capacité de débit suffisante.

Par exemple, supposons que vous ayez approvisionné 100 unités de capacité de lecture pour votre table DynamoDB. Cela vous permet de lire 409 600 octets par seconde (taille d'unité de capacité de lecture de 100 × 4 Ko). Supposons maintenant que la table contient 20 Go de données (21 474 836 480 octets) et que vous souhaitez utiliser l'instruction `SELECT` pour sélectionner toutes les données à l'aide de HiveQL. Vous pouvez estimer le temps nécessaire à l'exécution de la requête comme suit :

21 474 836 480 / 409 600 = 52 429 secondes = 14,56 heures

Dans ce scénario, la table DynamoDB est un goulot d'étranglement. Cela n'aide pas à ajouter des nœuds Amazon EMR, car le débit Hive est limité à seulement 409 600 octets par seconde. La seule façon de réduire le temps requis pour l'instruction `SELECT` consiste à augmenter la capacité de lecture allouée à la table DynamoDB.

Vous pouvez effectuer un calcul similaire pour estimer le temps nécessaire pour charger en bloc des données dans une table externe Hive mappée à une table DynamoDB. Déterminez le nombre total d'unités de capacité d'écriture nécessaires par élément (moins de 1 Ko = 1, 1 à 2 Ko = 2, etc.) et multipliez-le par le nombre d'éléments à charger. Vous obtiendrez le nombre d'unités de capacité d'écriture requises. Divisez ce nombre par le nombre d'unités de capacité d'écriture allouées par seconde. Cela donne le nombre de secondes nécessaires charger la table.

Vous devez surveiller régulièrement les métriques CloudWatch pour votre table. Pour obtenir une vue d'ensemble rapide de la console DynamoDB, choisissez votre table, puis l'onglet Métriques. À partir de là, vous pouvez afficher les unités de capacité de lecture et d'écriture consommées, ainsi que les demandes de lecture et d'écriture qui ont été limitées.

Capacité de lecture

Amazon EMR gère la charge de demande sur votre table DynamoDB, selon les paramètres de débit approvisionné de la table. Cependant, si vous observez un nombre important de messages `ProvisionedThroughputExceeded` dans la sortie de la tâche, vous pouvez ajuster la vitesse de lecture par défaut. Pour ce faire, vous pouvez modifier la variable de configuration `dynamodb.throughput.read.percent`. Vous pouvez utiliser la commande `SET` pour définir cette variable à l'invite de commande Hive :

```
SET dynamodb.throughput.read.percent=1.0;
```

Cette variable est conservée uniquement pour la session Hive en cours. Si vous quittez Hive et y revenez ultérieurement, `dynamodb.throughput.read.percent` revient à sa valeur par défaut.

La valeur de `dynamodb.throughput.read.percent` peut être de `0.1` à `1.5`. `0.5` représentant la vitesse de lecture par défaut, Hive tente de consommer la moitié de la capacité de lecture de la table. Si vous augmentez la valeur au-delà de `0.5`, Hive augmente le débit de demandes. Si vous réduisez la valeur sous `0.5`, le débit de demandes de lecture baisse. La vitesse de lecture réelle dépend de facteurs tels que l'existence d'une distribution uniforme des clés dans la clé DynamoDB.

Si vous remarquez que Hive épuise fréquemment la capacité de lecture approvisionnée de la table, ou si vos demandes de lecture sont trop limitées, essayez de réduire la valeur

`dynamodb.throughput.read.percent` sous `0.5`. Si vous disposez d'une capacité de lecture suffisante dans la table et souhaitez des opérations HiveQL plus réactives, vous pouvez définir la valeur au-dessus de `0.5`.

Capacité d'écriture

Amazon EMR gère la charge de demande sur votre table DynamoDB, selon les paramètres de débit approvisionné de la table. Cependant, si vous observez un nombre important de messages `ProvisionedThroughputExceeded` dans la sortie de la tâche, vous pouvez ajuster la vitesse d'écriture par défaut. Pour ce faire, vous pouvez modifier la variable de configuration `dynamodb.throughput.write.percent`. Vous pouvez utiliser la commande `SET` pour définir cette variable à l'invite de commande Hive :

```
SET dynamodb.throughput.write.percent=1.0;
```

Cette variable est conservée uniquement pour la session Hive en cours. Si vous quittez Hive et y revenez ultérieurement, `dynamodb.throughput.write.percent` revient à sa valeur par défaut.

La valeur de `dynamodb.throughput.write.percent` peut être de `0.1` à `1.5`. `0.5` représentant la vitesse d'écriture par défaut, Hive tente de consommer la moitié de la capacité d'écriture de la table. Si vous augmentez la valeur au-delà de `0.5`, Hive augmente le débit de demandes. Si vous réduisez la valeur sous `0.5`, le débit de demandes d'écriture baisse. La vitesse d'écriture réelle dépend de facteurs tels que l'existence d'une distribution uniforme des clés dans la clé DynamoDB.

Si vous remarquez que Hive épuise fréquemment la capacité d'écriture approvisionnée de la table, ou si vos demandes d'écriture sont trop limitées, essayez de réduire la valeur `dynamodb.throughput.write.percent` sous `0.5`. Si vous disposez d'une capacité d'écriture suffisante dans la table et souhaitez des opérations HiveQL plus réactives, vous pouvez définir la valeur au-dessus de `0.5`.

Lorsque vous écrivez des données sur DynamoDB à l'aide de Hive, assurez-vous que le nombre d'unités de capacité d'écriture est supérieur au nombre de mappeurs dans le cluster. Par exemple, envisagez un cluster Amazon EMR composé de 10 nœuds m1.xlarge. Le type de nœud m1.xlarge fournissant 8 tâches de mappage, le cluster compterait un total de 80 mappeurs (10 x 8). Si votre table DynamoDB possède moins de 80 unités de capacité d'écriture, une opération d'écriture Hive peut consommer tout le débit d'écriture de cette table.

Pour déterminer le nombre de mappeurs pour les types de nœuds Amazon EMR, consultez Configuration de la tâche dans le Manuel du développeur Amazon EMR.

Pour plus d'informations sur les mappeurs, consultez Ajustement des mappeurs.

Ajustement des mappeurs

Lorsque Hive lance une tâche Hadoop, celle-ci est traitée par une ou plusieurs tâches de mappage. En supposant que votre table DynamoDB dispose d'une capacité de débit suffisante, vous pouvez modifier le nombre de mappeurs dans le cluster afin d'améliorer les performances.

> **Note**
>
> Le nombre de tâches de mappage utilisées dans une tâche Hadoop est influencé par les fractionnements d'entrée où Hadoop subdivise les données en blocs logiques. Si Hadoop n'effectue pas suffisamment de fractionnements d'entrée, il se peut que vos opérations d'écriture ne puissent pas consommer tout le débit d'écriture disponible dans la table DynamoDB.

Augmentation du nombre de mappeurs

Chaque mappeur dans un Amazon EMR dispose d'un taux de lecture maximum de 1 Mio par seconde. Le nombre de mappeurs dans un cluster dépend de la taille des nœuds de votre cluster (pour plus d'informations sur la taille des nœuds et le nombre de mappeurs par nœud, consultez Configuration de la tâche dans le Manuel du développeur Amazon EMR).

Si votre table DynamoDB dispose d'une capacité de débit suffisante pour les lectures, vous pouvez essayer d'augmenter le nombre de mappeurs en effectuant l'une des opérations suivantes :

- Augmenter la taille des nœuds de votre cluster. Par exemple, si votre cluster utilise des nœuds m1.large (trois mappeurs par nœud), vous pouvez essayer d'opérer un mise à niveau vers des nœuds m1.xlarge (huit mappers par nœud).
- Augmenter le nombre de nœuds de votre cluster. Par exemple, si vous avez un cluster de trois nœuds m1.xlarge, vous disposez d'un total de 24 mappeurs. Si vous doublez la taille du cluster, avec le même type de nœud, vous disposez de 48 mappeurs.

Vous pouvez utiliser l'AWS Management Console pour gérer la taille ou le nombre des nœuds dans votre cluster (il se peut que vous deviez redémarrer le cluster pour que ces modifications prennent effet).

Un autre moyen d'augmenter le nombre de mappeurs consiste à modifier le paramètre de configuration Hadoop `mapred.tasktracker.map.tasks.maximum`. (comme il s'agit d'un paramètre Hadoop et non Hive, vous ne pouvez pas le modifier de manière interactive à partir de l'invite de commande). Si vous augmentez la valeur de `mapred.tasktracker.map.tasks.maximum`, vous pouvez augmenter le nombre de mappeurs sans augmenter la taille ou le nombre de nœuds. Cependant, il est possible que les nœuds de cluster manquent de mémoire si vous définissez une valeur trop élevée.

Vous définissez la valeur de `mapred.tasktracker.map.tasks.maximum` en tant qu'action d'amorçage lorsque vous lancez votre cluster Amazon EMR pour la première fois. Pour plus d'informations, consultez (Facultatif) Création d'actions d'amorçage pour installer des logiciels supplémentaires dans le Guide de gestion Amazon EMR.

Réduction du nombre de mappeurs

Si vous utilisez l'instruction `SELECT` pour sélectionner des données d'une table Hive externe qui mappe à DynamoDB, la tâche Hadoop peut utiliser autant de tâches que nécessaire, jusqu'au nombre maximum de mappeurs dans le cluster. Dans ce scénario, il est possible qu'une requête Hive de longue durée consomme toute la capacité de lecture approvisionnée de la table DynamoDB, ce qui a un impact négatif sur d'autres utilisateurs.

Vous pouvez utiliser le paramètre `dynamodb.max.map.tasks` afin de définir une limite supérieure pour les tâches de mappage :

```
SET dynamodb.max.map.tasks=1
```

Cette valeur doit être supérieure ou égale à 1. Lorsque Hive traite votre requête, la tâche Hadoop qui en résulte ne dépasse pas la capacité `dynamodb.max.map.tasks` lors de la lecture de la table DynamoDB.

Rubriques supplémentaires

Voici d'autres façons d'ajuster des applications qui utilisent Hive pour accéder à DynamoDB.

Retry duration (Durée de la nouvelle tentative)

Par défaut, Hive ré-exécute une tâche Hadoop qui n'a pas renvoyé de résultats de DynamoDB dans les deux minutes. Vous pouvez ajuster cet intervalle en modifiant le paramètre `dynamodb.retry.duration` :

```
SET dynamodb.retry.duration=2;
```

La valeur doit être un entier différent de zéro représentant le nombre de minutes de l'intervalle de nouvelle tentative. La valeur par défaut de `dynamodb.retry.duration` est 2 (minutes).

Demandes de données en parallèle

Plusieurs demandes de données adressées pas plusieurs utilisateurs ou plusieurs applications à une table unique peuvent réduire le débit de lecture approvisionné et ralentir les performances.

Durée du processus

La cohérence des données dans DynamoDB dépend de l'ordre des opérations de lecture et d'écriture sur chaque nœud. Quand une requête Hive est en cours, une autre application peut charger de nouvelles données dans la table DynamoDB, voire modifier ou supprimer des données existantes. Dans ce cas, les résultats de la requête Hive peuvent ne pas tenir compte des modifications apportées aux données pendant l'exécution de la requête.

Durée de la demande

La planification des requêtes Hive qui accèdent à une table DynamoDB à un moment où la demande sur celle-ci est inférieure a pour effet d'améliorer les performances. Par exemple, si la plupart des utilisateurs de votre application vivent à San Francisco, vous pouvez choisir d'exporter les données quotidiennes à 4h00 PST, lorsque la majorité des utilisateurs sont endormis et ne mettent pas à jour d'enregistrements dans votre base de données DynamoDB.

Intégration de DynamoDB à Amazon S3

Les fonctionnalités d'importation et d'exportation d'Amazon DynamoDB constituent un moyen simple et efficace de déplacer des données entre les tables Amazon S3 et DynamoDB sans écrire de code.

Les fonctionnalités d'importation et d'exportation de DynamoDB vous aident à déplacer, transformer et copier des comptes de table DynamoDB. Vous pouvez importer à partir de vos sources S3, exporter les données de vos tables DynamoDB vers Amazon S3 et utiliser des services AWS tels qu'Athena, Amazon SageMaker AI et AWS Lake Formation pour analyser vos données et en extraire des informations exploitables. Vous pouvez également importer des données directement dans de nouvelles tables DynamoDB pour créer de nouvelles applications avec des performances à l'échelle de quelques millisecondes, faciliter le partage des données entre les tables et les comptes et simplifier vos plans de reprise après sinistre et de continuité d'activité.

Rubriques

- Fonctionnement de l'importation de données vers DynamoDB depuis Amazon S3
- Fonctionnement de l'exportation de données DynamoDB vers Amazon S3

Fonctionnement de l'importation de données vers DynamoDB depuis Amazon S3

Pour importer des données dans DynamoDB, celles-ci doivent se trouver dans un compartiment Amazon S3 au format CSV, JSON DynamoDB ou Amazon Ion. Les données peuvent être compressées au format ZSTD ou GZIP, ou peuvent être importées directement sous une forme non compressée. Les données sources peuvent être un seul objet Amazon S3 ou plusieurs objets Amazon S3 qui utilisent le même préfixe.

Vos données seront importées dans une nouvelle table DynamoDB, qui sera créée lorsque vous lancerez la demande d'importation. Vous pouvez créer cette table avec des index secondaires, puis interroger et mettre à jour vos données dans tous les index primaires et secondaires dès que l'importation est terminée. Vous pouvez également ajouter un réplica de table globale une fois l'importation terminée.

> **Note**
>
> Au cours du processus d'importation Amazon S3, DynamoDB crée une nouvelle table cible pour l'importation. L'importation dans des tables existantes n'est actuellement pas prise en charge par cette fonction.

L'importation depuis Amazon S3 ne consomme pas de capacité d'écriture sur la nouvelle table, vous n'avez donc pas besoin de prévoir de capacité supplémentaire pour importer des données dans DynamoDB. La tarification de l'importation de données est basée sur la taille non compressée des données sources dans Amazon S3, qui est traitée à la suite de l'importation. Les éléments traités mais qui ne parviennent pas à être chargés dans la table en raison d'un formatage ou d'autres incohérences dans les données sources sont également facturés dans le cadre du processus d'importation. Pour plus de détails sur la tarification, consultez Tarification Amazon DynamoDB.

Vous pouvez importer des données depuis un compartiment S3 appartenant à un autre compte si vous disposez des autorisations appropriées pour lire dans ce compartiment. La nouvelle table

peut également se trouver dans une région différente de celle du compartiment Amazon S3 source. Pour de plus amples informations, veuillez consulter Configuration et autorisations d'Amazon Simple Storage Service.

La durée des importations est directement liée aux caractéristiques de vos données dans Amazon S3. Cela inclut la taille des données, le format des données, le schéma de compression, l'uniformité de la distribution des données, le nombre d'objets Amazon S3 et d'autres variables connexes. Plus particulièrement, les ensembles de données dont les clés sont réparties uniformément seront plus rapides à importer que les ensembles de données asymétriques. Par exemple, si la clé de votre index secondaire utilise le mois de l'année pour effectuer le partitionnement, mais que toutes vos données sont du mois de décembre, l'importation de ces données peut prendre beaucoup plus de temps.

Les attributs associés aux clés doivent être uniques dans la table de base. Si certaines clés ne sont pas uniques, l'importation remplacera les éléments associés jusqu'à ce que seul le dernier remplacement soit conservé. Par exemple, si la clé primaire est le mois et que plusieurs éléments sont définis sur le mois de septembre, chaque nouvel élément remplacera les éléments précédemment écrits et un seul élément avec la clé primaire « mois » définie sur septembre sera conservé. Dans ce cas, le nombre d'éléments traités dans la description de la table d'importation ne correspondra pas au nombre d'éléments de la table cible.

AWS CloudTrail enregistre toutes les actions de console et d'API pour l'importation de tables. Pour de plus amples informations, veuillez consulter Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail.

La vidéo suivante explique comment importer directement des données depuis Amazon S3 dans DynamoDB.

Importer des données depuis Amazon S3

Rubriques

- Demande d'importation de table dans DynamoDB
- Formats d'importation Amazon S3 pour DynamoDB
- Quotas de format d'importation et validation
- Bonnes pratiques pour importer des données depuis Amazon S3 vers DynamoDB

Demande d’importation de table dans DynamoDB

L’importation DynamoDB vous permet d’importer depuis un compartiment Amazon S3 versune nouvelle table DynamoDB. Vous pouvez demander une importation de table à l'aide de la console DynamoDB, de la CLI CloudFormationou de l'API DynamoDB.

Si vous souhaitez utiliser le AWS CLI, vous devez d'abord le configurer. Pour de plus amples informations, veuillez consulter Accès à DynamoDB.

> **Note**
>
> - La fonctionnalité Import Table interagit avec plusieurs AWS services différents tels qu'Amazon S3 et CloudWatch. Avant de commencer une importation, assurez-vous que l'utilisateur ou le rôle qui invoque l'importation APIs est autorisé à accéder à tous les services et ressources dont dépend la fonctionnalité.
> - Ne modifiez pas les objets Amazon S3 pendant que l’importation est en cours, car cela peut entraîner l’échec ou l’annulation de l’opération.
>
> Pour plus d’informations sur les erreurs et le dépannage, consultez Quotas de format d’importation et validation

Rubriques

- Configuration des autorisations IAM
- Demande d'exportation à l'aide de la AWS Management Console
- Pour en savoir plus sur les importations passées, consultez le AWS Management Console
- Demande d'importation à l'aide du AWS CLI
- Pour en savoir plus sur les importations passées, consultez le AWS CLI

Configuration des autorisations IAM

Vous pouvez importer des données à partir d’un compartiment Amazon S3 dans lequel vous êtes autorisé à lire. Le compartiment de destination ne doit pas nécessairement se trouver dans la même région ou avoir le même propriétaire que la table source. Votre Gestion des identités et des accès AWS(IAM) doit inclure les actions pertinentes sur le compartiment Amazon S3 source, ainsi que

les CloudWatch autorisations requises pour fournir des informations de débogage. Un exemple de stratégie est illustré ci-dessous.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
```

```
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
      "Resource": "*"
    }
  ]
}
```

Autorisations Amazon S3

Lorsque vous lancez une importation sur une source de compartiment Amazon S3 appartenant à un autre compte, assurez-vous que le rôle ou l'utilisateur a accès aux objets Amazon S3. Vous pouvez vérifier cela en exécutant la commande `GetObject` d'Amazon S3 et en utilisant les informations d'identification. Lors de l'utilisation de l'API, le paramètre propriétaire du compartiment Amazon S3 est défini par défaut sur l'ID de compte de l'utilisateur actuel. Pour les importations entre comptes, assurez-vous que ce paramètre est correctement renseigné avec l'ID de compte du propriétaire du compartiment. Le code suivant est un exemple de politique de compartiment Amazon S3 dans le compte source.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Key Management Service

Lorsque vous créez la nouvelle table à importer, si vous sélectionnez une clé de chiffrement au repos qui n'appartient pas à DynamoDB, vous devez fournir les autorisations requises pour exploiter une table DynamoDB chiffrée àAWS KMS l'aide de clés gérées par le client. Pour plus d'informations, voir Autoriser l'utilisation de votre AWS KMS clé. Si les objets Amazon S3 sont chiffrés avec le chiffrement KMS côté serveur (SSE-KMS), assurez-vous que le rôle ou l'utilisateur à l'origine de l'importation a accès au déchiffrement à l'aide de la clé.AWS KMS Cette fonction ne prend pas en charge les objets Amazon S3 chiffrés avec des clés de chiffrement fournies par le client (SSE-C).

CloudWatch autorisations

Le rôle ou l'utilisateur qui lance l'importation aura besoin des autorisations de création et de gestion pour le groupe de journaux et les flux de journaux associés à l'importation.

Demande d'exportation à l'aide de la AWS Management Console

L'exemple suivant montre comment utiliser la console DynamoDB pour importer une table existante nommée `MusicCollection`.

Pour demander une importation de table

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation sur le côté gauche de la console, choisissez Importer depuis S3.
3. Sur la page qui s'affiche, sélectionnez Import from S3 (Importer depuis S3).
4. ChoisissezImport from S3 (Importer depuis S3).
5. Dans URL source S3, saisissez l'URL source d'Amazon S3.

   Si le compartiment source vous appartient, choisissez Parcourir S3 pour le rechercher. Vous pouvez également saisir l'URL du compartiment au format suivant : `s3://bucket/prefix`. Le `prefix` est un préfixe de clé Amazon S3. Il s'agit du nom de l'objet Amazon S3 que vous souhaitez importer ou du préfixe de clé partagé par tous les objets Amazon S3 que vous souhaitez importer.

   > **Note**
   >
   > Vous ne pouvez pas utiliser le même préfixe que celui de votre demande d'exportation DynamoDB. La fonctionnalité d'exportation crée une structure de dossiers et des fichiers

manifestes pour toutes les exportations. Si vous utilisez le même emplacement Amazon S3, une erreur se produit.
Dirigez plutôt l'importation vers le dossier qui contient les données de cette exportation spécifique. Dans ce cas, le format du chemin correct est `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`, où `XXXXXXXX-XXXXXX` est l'ID d'exportation. Vous pouvez trouver l'ID d'exportation dans l'ARN d'exportation, qui a le format suivant : `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`. Par exemple, `arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/export/01234567890123-a1b2c3d4`.

6. Spécifiez si vous s êtes lePropriétaire du compartiment S3. Si le compartiment source appartient à un autre compte, sélectionnez Un AWS compte différent. Ensuite, entrez l'ID de compte du propriétaire du compartiment.
7. Sous Import file compression (Importer la compression de fichier), sélectionnez No compression (Aucune compression), GZIP ou ZSTD en fonction de vos besoins.
8. Sélectionnez le format de fichier d'importation approprié. Les options sont JSON DynamoDB, Amazon Ion ou CSV. Si vous sélectionnez CSV, deux options supplémentaires s'offrent à vous : CSV header (En-tête CSV) et CSV delimiter character (Caractère délimiteur CSV).

   Pour CSV header (En-tête CSV), choisissez si l'en-tête doit être extrait de la première ligne du fichier ou s'il doit être personnalisé. Si vous sélectionnez Customize your headers (Personnalisez vos en-têtes), vous pouvez spécifier les valeurs d'en-tête que vous souhaitez importer. Les en-têtes CSV spécifiés par cette méthode respectent la casse et sont censés contenir les clés de la table cible.

   Pour CSV delimiter character (Séparateur CSV), vous définissez le caractère qui séparera les éléments. La virgule est sélectionnée par défaut. Si vous sélectionnez Custom delimiter character (Séparateur personnalisé), le séparateur doit correspondre au modèle regex : `[,;:|\t ]`.
9. Sélectionnez le bouton Next (Suivant), puis sélectionnez les options de la nouvelle table qui sera créée pour stocker vos données.

> **Note**
>
> La clé primaire et la clé de tri doivent correspondre aux attributs du fichier, faute de quoi l'importation échouera. Les attributs sont sensibles à la casse.

10. Sélectionnez à nouveau Next (Suivant) pour vérifier vos options d'importation, puis cliquez sur Import (Importer) pour commencer la tâche d'importation. Vous verrez d'abord votre nouvelle table répertoriée dans les Tables avec le statut Creating (Création en cours). Pour le moment, la table n'est pas accessible.
11. Une fois l'importation terminée, le statut indique Active et vous pouvez commencer à utiliser la table.

Pour en savoir plus sur les importations passées, consultez le AWS Management Console

Vous pouvez trouver des informations sur les tâches d'importation que vous avez exécutées dans le passé en cliquant sur Import from S3 (Importer depuis S3) dans la barre latérale de navigation, puis en sélectionnant l'onglet Imports (Importations). Le panneau d'importation contient la liste de toutes les importations que vous avez effectuées au cours des 90 derniers jours. La sélection de l'ARN d'une tâche répertoriée sous l'onglet Imports (Importations) a pour effet d'afficher les informations relatives à cette importation, dont les paramètres de configuration avancés que vous avez choisis.

Demande d'importation à l'aide du AWS CLI

L'exemple suivant importe des données au format CSV d'un compartiment S3 appelé bucket avec un préfixe dans une nouvelle table appelée target-table.

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> **Note**
>
> Si vous choisissez de chiffrer votre importation à l'aide d'une clé protégée par AWS Key Management Service(AWS KMS), la clé doit se trouver dans la même région que le compartiment Amazon S3 de destination.

Pour en savoir plus sur les importations passées, consultez le AWS CLI

Vous pouvez trouver des informations sur des tâches d'importation que vous avez exécutées dans le passé à l'aide de la commande `list-imports`. Cette commande envoie la liste de toutes les importations que vous avez effectuées au cours des 90 derniers jours. Notez que, bien que les métadonnées de tâche d'importation expirent après 90 jours et que les tâches plus anciennes ne figurent plus dans cette liste, DynamoDB ne supprime aucun des objets de votre compartiment Amazon S3 ou de la table créée lors de l'importation.

```
aws dynamodb list-imports
```

Pour extraire des informations détaillées sur une tâche d'importation spécifique, dont ses paramètres de configuration avancés, utilisez la commande `describe-import`.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Formats d'importation Amazon S3 pour DynamoDB

DynamoDB peut importer des données dans trois formats : CSV, JSON DynamoDB et Amazon Ion.

Rubriques

- CSV
- Json DynamoDB
- Amazon Ion

CSV

Un fichier au format CSV se compose de plusieurs éléments délimités par des sauts de ligne. Par défaut, DynamoDB interprète la première ligne d'un fichier d'importation comme en-tête et s'attend

à ce que les colonnes soient délimitées par des virgules. Vous pouvez également définir des en-têtes qui seront appliqués, à condition qu'ils correspondent au nombre de colonnes du fichier. Si vous définissez des en-têtes explicitement, la première ligne du fichier sera importée en tant que valeurs.

> **Note**
>
> Lors de l'importation à partir de fichiers CSV, toutes les colonnes autres que la plage de hachage et les clés de votre table de base et des index secondaires sont importées en tant que chaînes DynamoDB.

Échapper les guillemets

Tous les guillemets présents dans le fichier CSV doivent être échappés. S'ils ne sont pas échappés, comme dans l'exemple suivant, l'importation échoue :

```
id,value
"123",Women's Full "Length" Dress
```

Cette même importation aboutit si les guillemets sont échappés par deux paires de guillemets :

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

Une fois que le texte a été correctement échappé et importé, il apparaît tel qu'il était dans le fichier CSV d'origine :

```
id,value
"123",Women's Full "Length" Dress
```

Json DynamoDB

Un ficher au format JSON DynamoDB peut comprendre plusieurs objets d'éléments. Chaque objet est au format JSON trié standard et des sauts de ligne sont utilisés en tant que délimiteurs d'éléments. En tant que fonction ajoutée, les exportations à partir d'un point dans le temps sont prises en charge en tant que source d'importation par défaut.

> **Note**
>
> Les nouvelles lignes sont utilisées comme délimiteurs d'éléments pour un fichier au format DynamoDB JSON et ne doivent pas être utilisées dans un objet d'élément.


```
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "333-3333333333"}, "Id": {"N": "103"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 103 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "444-444444444"}, "Id": {"N": "104"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 104 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "555-5555555555"}, "Id": {"N": "105"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 105 Title"}}}
```


Amazon Ion

Amazon Ion est un format de sérialisation de données hiérarchique riche, auto-descriptif et conçu pour répondre aux défis de développement rapide, de découplage et d'efficacité rencontrés lors de la conception d'architectures orientées service à grande échelle.

Lorsque vous importez des données au format Ion, les types de données Ion sont mappés à des types de données DynamoDB dans la nouvelle table DynamoDB.

N° de série	Conversion du type de données Ion en DynamoDB	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`

Nº de série	Conversion du type de données Ion en DynamoDB	B
5	`blob`	`Binary (B)`
6	`list (with type annotation $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)`	`Set (SS, NS, BS)`
7	`list`	`List`
8	`struct`	`Map`

Les éléments d'un fichier Ion sont délimités par des sauts de ligne. Chaque ligne commence par un marqueur de version Ion, suivi d'un élément au format Ion.

> **Note**
>
> Dans l'exemple suivant, un élément d'un fichier au format Ion a été mis en forme sur plusieurs lignes par souci de lisibilité.

```
$ion_1_0
[
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"333-3333333333",
      Id:103.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 103 Title"
    }
  },
  {
    Item:{
```

```
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"444-4444444444",
      Id:104.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 104 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"555-5555555555",
      Id:105.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 105 Title"
    }
  }
]
```

Quotas de format d'importation et validation

Quotas d'importation

L'importation vers DynamoDB depuis Amazon S3 peut prendre en charge jusqu'à 50 tâches d'importation simultanées avec une taille d'objet d'importation totale de 15 To à la fois dans les régions us-east-1, us-west-2, us-west-1. Dans toutes les autres régions, jusqu'à 50 tâches d'importation simultanées d'une taille totale de 1 To sont prises en charge. Chaque tâche d'importation peut prendre jusqu'à 50 000 objets Amazon S3 dans toutes les régions. Ces quotas par défaut sont appliqués à tous les comptes. Si vous pensez avoir besoin de modifier ces quotas, veuillez contacter votre équipe chargée de compte, qui examinera la situation au cas par cas. Pour plus d'informations sur les limites de DynamoDB, consultez Quotas de service.

Erreurs de validation

Au cours du processus d'importation, DynamoDB peut rencontrer des erreurs lors de l'analyse de vos données. Pour chaque erreur, DynamoDB émet un journal CloudWatch et tient un décompte du nombre total d'erreurs rencontrées. Si l'objet Amazon S3 lui-même est mal formé ou si son contenu ne peut pas former un élément DynamoDB, nous pouvons ignorer le traitement de la partie restante de l'objet.

> **Note**
>
> Si la source de données Amazon S3 contient plusieurs éléments qui partagent la même clé, ces éléments seront remplacés jusqu'à ce qu'il en reste un. Cela peut donner l'impression qu'un élément a été importé et que les autres ont été ignorés. Les éléments dupliqués seront remplacés dans un ordre aléatoire, ne seront pas comptabilisés comme des erreurs et ne seront pas transmis aux journaux CloudWatch.
> Une fois l'importation terminée, vous pouvez voir le nombre total d'éléments importés, le nombre total d'erreurs et le nombre total d'éléments traités. Pour un dépannage plus approfondi, vous pouvez également vérifier la taille totale des éléments importés et la taille totale des données traitées.

Il existe trois catégories d'erreurs d'importation : les erreurs de validation d'API, les erreurs de validation des données et les erreurs de configuration.

Erreurs de validation d'API

Les erreurs de validation d'API sont des erreurs au niveau des éléments provenant de l'API de synchronisation. Les causes courantes sont les problèmes d'autorisation, l'absence de paramètres requis et les échecs de validation des paramètres. Les détails sur les raisons de l'échec de l'appel d'API figurent dans les exceptions générées par la demande `ImportTable`.

Erreurs de validation

Des erreurs de validation des données peuvent se produire au niveau de l'article ou du fichier. Lors de l'importation, les éléments sont validés en fonction des règles DynamoDB avant d'être importés dans la table cible. Lorsqu'un article échoue à la validation et n'est pas importé, la tâche d'importation ignore cet élément et continue avec l'élément suivant. À la fin de la tâche, l'état de l'importation est défini sur FAILED (échec) avec un FailureCode, une ItemValidationError et le

message FailureMessage « Certains éléments ont échoué aux contrôles de validation et n'ont pas été importés. Consultez les journaux d'erreurs CloudWatch pour plus d'informations. »

Les causes courantes des erreurs de validation des données sont les objets non analysables, le format incorrect des objets (l'entrée spécifie DYNAMODB_JSON, mais l'objet n'est pas dans DYNAMODB_JSON) et la non-correspondance de schéma avec les clés de table source spécifiées.

Erreurs de configuration

Les erreurs de configuration sont généralement des erreurs de workflow dues à la validation des autorisations. Le flux de travail d'importation vérifie certaines autorisations après avoir accepté la demande. En cas de problème lors de l'appel de l'une des dépendances requises, comme Amazon S3 ou CloudWatch, le processus marque l'état de l'importation comme FAILED (échec). `failureCode` et `failureMessage` indiquent la raison de l'échec. Le cas échéant, le message d'échec contient également l'ID de demande que vous pouvez utiliser pour rechercher la raison de l'échec dans CloudTrail.

Les erreurs de configuration courantes incluent l'URL erronée pour le compartiment Amazon S3 et le fait de ne pas avoir l'autorisation d'accéder au compartiment Amazon S3, à CloudWatch Logs, et les clés AWS KMS utilisées pour déchiffrer l'objet Amazon S3. Pour de plus amples informations, veuillez consulter Utilisation des clés KMS et des clés de données.

Validation des objets Amazon S3 sources

Pour valider les objets S3 sources, réalisez les étapes suivantes.

1. Validez le format des données et le type de compression

   - Assurez-vous que tous les objets Amazon S3 correspondants sous le préfixe spécifié ont le même format (DYNAMODB_JSON, DYNAMODB_ION, CSV)
   - Assurez-vous que tous les objets Amazon S3 correspondants sous le préfixe spécifié sont compressés de la même manière (GZIP, ZSTD, NONE)

   > **Note**
   >
   > Les objets Amazon S3 n'ont pas besoin de l'extension correspondante (.csv/.json/.ion/.gz/.zstd, etc.) car le format d'entrée spécifié dans l'appel ImportTable a priorité.

2. Vérifiez que les données d'importation sont conformes au schéma de table souhaité

- Assurez-vous que chaque élément des données sources possède la clé primaire. Une clé de tri est facultative pour les importations.
- Assurez-vous que le type d'attribut associé à la clé primaire et à toute clé de tri correspond au type d'attribut de la table et du schéma GSI, comme spécifié dans les paramètres de création de table

Résolution des problèmes

CloudWatch Logs

Pour les tâches d'importation qui échouent, des messages d'erreur détaillés sont publiés dans les journaux CloudWatch. Pour accéder à ces journaux, récupérez d'abord ImportARN à partir de la sortie et describe-import à l'aide de cette commande :

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Exemple de sortie :

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
        "InputCompressionType": "NONE",
```

```
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

Récupérez le groupe de journaux et l'ID d'importation de la réponse ci-dessus et utilisez-les pour récupérer les journaux d'erreurs. L'ID d'importation est le dernier élément de chemin du champ `ImportArn`. Le nom du groupe de journaux est `/aws-dynamodb/imports`. Le nom du flux de journaux d'erreurs est `import-id/error`. Pour cet exemple, ce serait `01658528578619-c4d4e311/error`.

Il manque la clé pk dans l'élément

Si l'objet S3 source ne contient pas la clé primaire fournie en tant que paramètre, l'importation échouera. Par exemple, lorsque vous définissez la clé primaire pour l'importation en tant que nom de colonne « pk ».

```
aws dynamodb import-table –s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
```

```
            –input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

La colonne « pk » est absente de l'objet source `import-test.csv` qui a le contenu suivant :

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Cette importation échouera en raison d'une erreur de validation d'élément due à l'absence de clé primaire dans la source de données.

Exemple de journal des erreurs CloudWatch :

```
aws logs get-log-events –log-group-name /aws-dynamodb/imports –log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

Ce journal d'erreurs indique que « une ou plusieurs valeurs de paramètre n'étaient pas valides : il manquait la clé pk dans l'élément ». Comme cette tâche d'importation a échoué, la table « target-table » existe désormais et elle est vide car aucun élément n'a été importé. Le premier élément a été traité et la validation de l'objet a échoué.

Pour résoudre le problème, supprimez d'abord « target-table » si elle n'est plus nécessaire. Ensuite, utilisez un nom de colonne de clé primaire qui existe dans l'objet source ou mettez à jour les données source pour :

```
pk,title,artist,year_of_release
```

```
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

La table cible existe

Lorsque vous démarrez une tâche d'importation et que vous recevez une réponse comme suit :

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Pour corriger cette erreur, vous devez choisir un nom de table qui n'existe pas déjà et réessayer l'importation.

Le compartiment spécifié n'existe pas

Si le compartiment source n'existe pas, l'importation échouera et consignera les détails du message d'erreur dans Cloudwatch.

Exemple de description de l'importation :

```
aws dynamodb –endpoint-url $ENDPOINT describe-import –import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
```

```
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```


Le `FailureCode` est `S3NoSuchBucket`, avec le `FailureMessag` contenant des détails tels que l'identifiant de la demande et le service qui a généré l'erreur. Étant donné que l'erreur a été détectée avant l'importation des données dans la table, aucune nouvelle table DynamoDB n'est créée. Dans certains cas, lorsque ces erreurs se produisent après le début de l'importation des données, la table contenant des données partiellement importées est conservée.

Pour corriger cette erreur, assurez-vous que le compartiment Amazon S3 source existe, puis redémarrez le processus d'importation.

Bonnes pratiques pour importer des données depuis Amazon S3 vers DynamoDB

Les bonnes pratiques ci-dessous s'appliquent à l'importation de données depuis Amazon S3 vers DynamoDB

Non dépassement de la limite de 50 000 objets S3

Chaque tâche d'importation prend en charge 50 000 objets S3 au maximum. Si votre jeu de données contient plus de 50 000 objets, envisagez de les regrouper en objets de taille supérieure.

Éviter les objets S3 trop volumineux

Les objets S3 sont importés en parallèle. Le fait de disposer de nombreux objets S3 de taille moyenne permet une exécution parallèle sans surcharge excessive. Pour les éléments de moins de 1 Ko, envisagez de placer 4 000 000 d'éléments dans chaque objet S3. Si la taille moyenne de vos objets est plus grande, placez proportionnellement moins d'éléments dans chaque objet S3.

Rendre les données triées aléatoires

Si un objet S3 contient des données dans un ordre trié, il peut créer une partition dynamique. Dans cette situation, une partition reçoit toute l'activité, puis la partition suivante, etc. Les données triées sont définies en tant qu'éléments en séquence dans l'objet S3 qui seront écrits sur la même partition cible lors de l'importation. Une situation courante dans laquelle les données sont triées est un fichier CSV dans lequel les éléments sont triés par clé de partition, afin que les éléments répétés partagent la même clé de partition.

Pour éviter de créer une partition dynamique, nous vous recommandons de choisir un ordre aléatoire pour ces cas. Cela peut améliorer les performances, en répartissant les opérations d'écriture. Pour de plus amples informations, veuillez consulter Distribution efficace de l'activité d'écriture pendant le chargement de données dans DynamoDB.

Compresser les données pour maintenir la taille totale des objets S3 en dessous de la limite régionale

Dans le processus d'importation depuis S3, la taille totale des données d'objet S3 à importer est limitée. La limite est de 15 To dans les régions us-east-1, us-west-2 et eu-west-1, et 1 To dans toutes les autres Régions. La limite est basée sur les tailles brutes des objets S3.

La compression permet à un plus grand nombre de données brutes de respecter la limite. Si la compression à elle seule ne suffit pas à maintenir l'importation dans les limites, vous pouvez également contacter le Support Premium AWS pour obtenir une augmentation du quota.

Noter l'impact de la taille de l'élément sur les performances

Si la taille moyenne de vos éléments est très petite (inférieure à 200 octets), le processus d'importation peut prendre un peu plus de temps que pour les éléments de grande taille.

Ne modifiez pas les objets S3 pendant les importations actives

Assurez-vous que vos objets S3 source restent inchangés pendant qu'une opération d'importation est en cours. Si un objet S3 est modifié lors d'une importation, l'opération échouera avec le code d'erreur

`ObjectModifiedInS3DuringImport` et le message « L'objet S3 n'a pas pu être importé car il a été remplacé ».

Si vous rencontrez cette erreur, redémarrez l'opération d'importation avec une version stable de votre objet S3. Pour éviter ce problème, attendez que l'importation en cours soit terminée avant de modifier les fichiers source.

Envisager d’importer sans aucun index secondaire global

La durée d'une tâche d'importation peut dépendre de la présence d'un ou de plusieurs index secondaires globaux (GSIs). Si vous prévoyez de créer des index avec des clés de partition à faible cardinalité, vous pouvez accélérer l’importation si vous repoussez la création de l’index après la fin de la tâche d’importation (plutôt que de les inclure dans la tâche d’importation).

> **Note**
>
> La création d’un GSI lors de l’importation n’entraîne pas de frais d’écriture (contrairement à la création d’un GSI après l’importation).

Fonctionnement de l’exportation de données DynamoDB vers Amazon S3

L’exportation de DynamoDB vers S3 est une solution entièrement gérée permettant d’exporter à grande échelle vos données DynamoDB vers un compartiment Amazon S3. Grâce à l'exportation DynamoDB vers S3, vous pouvez exporter les données d'une table Amazon DynamoDB à tout moment au cours de point-in-time votre fenêtre de restauration (PITR) vers un compartiment Amazon S3. Pour pouvoir utiliser la fonctionnalité d’exportation, vous devez activer PITR sur votre table. Cette fonctionnalité vous permet d'effectuer des analyses et des requêtes complexes sur vos données à l'aide d'autres AWS services tels qu'Athena,AWS Glue Amazon SageMaker AI, Amazon EMR et.AWS Lake Formation

L’exportation de DynamoDB vers S3 vous permet d’exporter des données complètes et incrémentielles depuis votre table DynamoDB. Les exportations sont asynchrones, elles ne consomment pas d'unités de capacité de lecture (RCUs) et n'ont aucun impact sur les performances et la disponibilité des tables. Les formats de fichiers d’exportation pris en charge sont les formats JSON de DynamoDB et Amazon Ion. Vous pouvez également exporter des données vers un compartiment S3 appartenant à un autre AWS compte et vers une autre AWS région. Vos données sont toujours cryptées end-to-end.

Les exportations complètes DynamoDB sont facturées en fonction de la taille de la table DynamoDB (données de table et index secondaires locaux) à l'instant pour lequel l'exportation est effectuée. Les exportations incrémentielles DynamoDB sont facturées en fonction de la taille des données traitées à partir de vos sauvegardes continues pour la période concernée par l'exportation. L'exportation incrémentielle a une facturation minimale de 10 Mo. Des frais supplémentaires s'appliquent pour le stockage des données exportées dans Amazon S3 et pour les demandes PUT adressées à votre compartiment Amazon S3. Pour en savoir plus sur ces frais, consultez Tarification Amazon DynamoDB et Tarification Amazon S3.

Pour plus de détails sur les quotas de service, consultez Exportation de table vers Amazon S3..

Rubriques

- Demande d'exportation de table dans DynamoDB
- Format de sortie d'exportation de table DynamoDB

Demande d'exportation de table dans DynamoDB

Les exportations de tables DynamoDB vous permettent d'exporter les données des tables vers un compartiment Amazon S3, ce qui vous permet d'effectuer des analyses et des requêtes complexes sur vos données à l'aide AWS d'autres services tels qu'Athena,AWS Glue Amazon AI SageMaker , Amazon EMR et.AWS Lake Formation Vous pouvez demander une exportation de table à l'aide de l'AWS Management ConsoleAWS CLI API DynamoDB ou de l'API DynamoDB.

> **Note**
>
> Le demandeur paie les compartiments Amazon S3 qui ne sont pas pris en charge.

DynamoDB prend en charge à la fois l'exportation complète et l'exportation incrémentielle :

- Avec les exportations complètes, vous pouvez exporter un instantané complet de votre table à tout moment pendant la fenêtre de point-in-time restauration (PITR) vers votre compartiment Amazon S3.
- Avec les exportations incrémentielles, vous pouvez exporter vers votre compartiment Amazon S3 les données de votre table DynamoDB qui ont été modifiées, mises à jour ou supprimées au cours d'une période spécifiée dans votre fenêtre PITR.

Rubriques

- Conditions préalables
- Demande d’exportation à l’aide de la AWS Management Console
- Obtenir des informations sur les exportations passées dans le AWS Management Console
- Demande d'exportation à l'aide de l'AWS CLI
- Obtenir des informations sur les exportations passées dans le AWS CLI
- Demande d’une exportation à l’aide du kit SDK AWS
- Obtenir des informations sur les exportations passées à l'aide du AWS SDK

Conditions préalables

Activer PITR

Pour utiliser la fonctionnalité d’exportation vers S3, vous devez activer la PITR sur votre table. Pour plus de détails sur l'activation du PITR, voir P oint-in-time recovery. Si vous demandez l'exportation d'une table pour laquelle le PITR n'est pas activé, votre demande échouera avec un message d'exception : « Une erreur s'est produite (PointInTimeRecoveryUnavailableException) lors de l'appel de l'`ExportTableToPointInTime`opération : la restauration instantanée n'est pas activée pour la table 'my-dynamodb-table». Vous ne pouvez demander et exporter qu’à partir d’un point dans le temps correspondant à la PITR `RecoveryPeriodInDays` configurée.

Configuration des autorisations S3

Vous pouvez exporter vos données de table vers n’importe quel compartiment Amazon S3 dans lequel vous êtes autorisé à écrire. Il n'est pas nécessaire que le compartiment de destination se trouve dans la même AWS région ou qu'il ait le même propriétaire que le propriétaire de la table source. Votre politique Gestion des identités et des accès AWS(IAM) doit vous permettre d'effectuer des actions S3 (`s3:AbortMultipartUpload`,`s3:PutObject`, et`s3:PutObjectAcl`) et l'action d'exportation DynamoDB (). `dynamodb:ExportTableToPointInTime` Voici un exemple de stratégie qui accorde à vos utilisateurs les autorisations nécessaires pour effectuer des exportations vers un compartiment S3.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Si vous devez écrire dans un compartiment S3 qui se trouve dans un autre compte ou si vous n'êtes pas autorisé à y écrire, le propriétaire du compartiment S3 doit ajouter une stratégie de compartiment pour vous permettre d'exporter de DynamoDB vers ce compartiment. Voici un exemple de stratégie sur le compartiment S3 cible.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
```

```
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

La révocation de ces autorisations pendant une exportation entraînera la création de fichiers partiels.

> **Note**
>
> Si la table ou le compartiment vers lequel vous effectuez l'exportation est chiffré à l'aide de clés gérées par le client, les politiques associées à cette clé KMS doivent accorder à DynamoDB l'autorisation de l'utiliser. Cette autorisation est accordée par le biais de l'IAM User/Role qui déclenche la tâche d'exportation. Pour plus d'informations sur le chiffrement, notamment sur les bonnes pratiques en la matière, consultez Comment DynamoDB utilise AWS KMS et Using a custom KMS key (Utilisation d'une clé KMS personnalisée).

Demande d'exportation à l'aide de la AWS Management Console

L'exemple suivant montre comment utiliser la console DynamoDB pour exporter une table existante nommée `MusicCollection`.

> **Note**
>
> Cette procédure suppose que vous avez activé point-in-time la restauration. Pour l'activer pour la `MusicCollection` table, dans l'onglet Vue d'ensemble de la table, dans la section Détails de la table, choisissez Activer pour la oint-in-timerestauration P.

Pour demander une exportation de table

1. Connectez-vous à la console DynamoDB AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/dynamodb/
2. Dans le panneau de navigation sur le côté gauche de la console, choisissez Exportations vers S3.
3. Sélectionnez le bouton Exporter vers S3.
4. Choisissez une table source et un compartiment S3 de destination. Si le compartiment de destination appartient à votre compte, vous pouvez utiliser le bouton Browse S3 (Parcourir S3)

pour le trouver. Sinon, saisissez l'URL du compartiment à l'aide du `s3://bucketname/prefix format.`. Le **`prefix`** est un dossier facultatif qui vous permet de garder votre compartiment de destination organisé.

5. Choisissez Exportation complète ou Exportation incrémentielle. Une exportation complète sort l'instantané complet de votre table telle qu'elle était à l'instant spécifié. Une exportation incrémentielle sort les modifications apportées à votre table au cours de la période d'exportation spécifiée. Votre sortie est compactée de telle sorte qu'elle contienne uniquement l'état final de l'élément de la période d'exportation. L'élément n'apparaît qu'une seule fois dans l'exportation, même s'il a été mis à jour plusieurs fois au cours de la même période d'exportation.

   Full export

   1. Sélectionnez l'instant précis à partir duquel vous voulez que l'instantané complet de la table soit exporté. Cela peut être n'importe quel instant de la fenêtre PITR. Vous pouvez également sélectionner Heure actuelle pour exporter l'instantané le plus récent.
   2. Pour Format de fichier exporté, choisissez entre JSON DynamoDB et Amazon Ion. Par défaut, votre table sera exportée au format JSON DynamoDB à partir de la dernière heure restaurable dans la fenêtre de restauration à un instant dans le passé, et chiffrée à l'aide d'une clé Amazon S3 (SSE-S3). Vous pouvez modifier ces paramètres d'exportation si nécessaire.

      > **Note**
      >
      > Si vous choisissez de chiffrer votre exportation à l'aide d'une clé protégée par AWS Key Management Service(AWS KMS), la clé doit se trouver dans la même région que le compartiment S3 de destination.

   Incremental export

   1. Sélectionnez la Période d'exportation pour laquelle vous souhaitez exporter les données incrémentielles. Choisissez une heure de début dans la fenêtre PITR. La durée de la période d'exportation doit être au minimum de 15 minutes et ne pas dépasser 24 heures. L'heure de début de la période d'exportation est incluse et l'heure de fin est exclue.
   2. Choisissez entre le Mode absolu et le Mode relatif.

      a. Le Mode absolu exporte les données incrémentielles correspondant à la période que vous spécifiez.

b. Le Mode relatif exporte les données incrémentielles pour une période d'exportation relative à la date de soumission de votre tâche d'exportation.

3. Pour Format de fichier exporté, choisissez entre JSON DynamoDB et Amazon Ion. Par défaut, votre table sera exportée au format JSON DynamoDB à partir de la dernière heure restaurable dans la fenêtre de restauration à un instant dans le passé, et chiffrée à l'aide d'une clé Amazon S3 (SSE-S3). Vous pouvez modifier ces paramètres d'exportation si nécessaire.

> **Note**
>
> Si vous choisissez de chiffrer votre exportation à l'aide d'une clé protégée par AWS Key Management Service(AWS KMS), la clé doit se trouver dans la même région que le compartiment S3 de destination.

4. Pour Type de vue d'exportation, sélectionnez Ancienne et nouvelle images ou Nouvelles images uniquement. Nouvelle image indique l'état le plus récent de l'élément. Ancienne image indique l'état de l'élément juste avant les « date et heure de début » spécifiées. Le paramètre par défaut est Ancienne et nouvelle images. Pour en savoir plus sur les nouvelles et anciennes images, consultez Sortie d'une exportation incrémentielle.

6. Choisissez Exporter pour commencer.

Les données exportées ne sont pas cohérentes du point de vue des transactions. Vos opérations de transaction peuvent être réparties sur deux sorties d'exportation. Il est possible qu'un sous-ensemble d'éléments modifiés par une opération de transaction apparaisse dans l'exportation, mais qu'un autre sous-ensemble de modifications relevant de la même transaction n'apparaisse pas dans la même demande d'exportation. Cependant, les exportations sont cohérentes au final. Si une transaction est divisée lors d'une exportation, la transaction restante sera intégrée à votre prochaine exportation contiguë, sans doublons. Les périodes de temps utilisées pour les exportations reposent sur une horloge système interne et peuvent varier d'une minute par rapport à l'horloge locale de votre application.

Obtenir des informations sur les exportations passées dans le AWS Management Console

Vous pouvez trouver des informations sur les tâches d'exportation que vous avez exécutées antérieurement en choisissant la section Exportations vers S3 de la barre latérale de navigation. Cette section contient la liste de toutes les exportations que vous avez créées au cours des 90 derniers jours. Sélectionnez l'ARN d'une tâche répertoriée sous l'onglet Exportations pour extraire

les informations relatives à cette exportation, dont les paramètres de configuration avancés que vous avez choisis. Notez que, bien que les métadonnées de tâche d'exportation expirent après 90 jours et que les tâches plus anciennes ne figurent plus dans cette liste, les objets restent dans votre compartiment S3 aussi longtemps que leurs politiques de compartiment le permettent. DynamoDB ne supprime jamais aucun des objets qu'il crée dans votre compartiment S3 lors d'une exportation.

Demande d'exportation à l'aide de l'AWS CLI

L'exemple suivant montre comment utiliser le AWS CLI pour exporter une table existante nommée `MusicCollection` vers un compartiment S3 appelé`ddb-export-musiccollection`.

> **Note**
>
> Cette procédure suppose que vous avez activé point-in-time la restauration. Pour l'activer pour la table `MusicCollection`, exécutez la commande suivante.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

La commande suivante exporte `MusicCollection` vers un compartiment S3 nommé `ddb-export-musiccollection-9012345678` en y ajoutant le préfixe `2020-Nov`. Les données de la table sont exportées au format JSON DynamoDB telles qu'elles étaient à un moment spécifique de la fenêtre de restauration à un instant dans le passé, et chiffrées à l'aide d'une clé Amazon S3 (SSE-S3).

> **Note**
>
> Si vous demandez une exportation de table entre comptes, veillez à inclure l'option `--s3-bucket-owner`.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
```

```
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Incremental export

La commande suivante effectue une exportation incrémentielle en fournissant un nouveau `--export-type` et une `--incremental-export-specification`. Remplacez ce qui apparaît en italique par vos propres valeurs. Les temps sont exprimés en secondes depuis l'epoch.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

> **Note**
>
> Si vous choisissez de chiffrer votre exportation à l'aide d'une clé protégée par AWS Key Management Service(AWS KMS), la clé doit se trouver dans la même région que le compartiment S3 de destination.

Obtenir des informations sur les exportations passées dans le AWS CLI

Vous pouvez trouver des informations sur des demandes d'exportation que vous avez exécutées antérieurement à l'aide de la commande `list-exports`. Cette commande envoie la liste de toutes les exportations que vous avez effectuées au cours des 90 derniers jours. Notez que, bien que les métadonnées de tâche d'exportation expirent après 90 jours et que la commande `list-exports` ne renvoie plus les tâches plus anciennes, les objets restent dans votre compartiment S3 aussi longtemps que leurs politiques de compartiment le permettent. DynamoDB ne supprime jamais aucun des objets qu'il crée dans votre compartiment S3 lors d'une exportation.

Les exportations ont le statut `PENDING` tant qu'elles n'ont pas abouti ou échoué. En cas de réussite, le statut passe à `COMPLETED`. En cas d'échec, le statut passe à `FAILED` avec un `failure_message` et un `failure_reason`.

Dans l'exemple suivant, nous utilisons le paramètre factultatif `table-arn` pour répertorier uniquement les exportations d'une table spécifique.

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Pour extraire des informations détaillées sur une tâche d'exportation spécifique, dont ses paramètres de configuration avancés, utilisez la commande `describe-export`.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

Demande d'une exportation à l'aide du kit SDK AWS

Utilisez ces extraits de code pour demander une exportation de table à l'aide du AWS SDK de votre choix.

Python

Exportation complète

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Exportation incrémentielle

```
import boto3
```

```
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Obtenir des informations sur les exportations passées à l'aide du AWS SDK

Utilisez ces extraits de code pour obtenir des informations sur les anciennes exportations de tables à l'aide du AWS SDK de votre choix.

Python

List

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

Décrire

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Format de sortie d'exportation de table DynamoDB

Une exportation de table DynamoDB comprend des fichiers manifeste en plus des fichiers contenant les données de votre table. Ces fichiers sont enregistrés dans le compartiment Amazon S3 que vous spécifiez dans votre demande d'exportation. Les sections suivantes décrivent le format et le contenu de chaque objet en sortie.

Sortie d'une exportation complète

Fichiers manifestes

DynamoDB crée des fichiers manifeste, ainsi que leurs fichiers de somme de contrôle, dans le compartiment S3 spécifié pour chaque demande d'exportation.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Vous choisissez un **`export-prefix`** lorsque vous demandez une exportation de table. Cela vous aide à organiser les fichiers dans le compartiment S3 de destination. L'**`ExportId`** est un jeton unique généré par le service qui vise à éviter que plusieurs exportations vers un même compartiment S3 et `export-prefix` ne se remplacent les unes les autres.

L'exportation crée au moins un fichier par partition. Pour les partitions vides, votre demande d'exportation crée un fichier vide. Tous les éléments de chaque fichier proviennent de l'espace de clés haché de cette partition particulière.

> **Note**
>
> DynamoDB crée également un fichier vide nommé `_started` dans le même répertoire que les fichiers manifestes. Ce fichier vérifie que le compartiment de destination est accessible en écriture et que l'exportation a commencé. Il peut être supprimé en toute sécurité.

Manifeste de récapitulatif

Le fichier `manifest-summary.json` contient des informations récapitulatives sur la tâche d'exportation. Cela vous permet de savoir quels fichiers de données du dossier de données partagé sont associés à cette exportation. Son format est le suivant :

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

Manifeste de fichiers

Le fichier `manifest-files.json` contient des informations sur les fichiers contenant les données de votre table exportée. Le fichier étant au format JSON Lines, de nouvelles lignes sont utilisées

comme délimiteurs d'éléments. Dans l'exemple suivant, les détails relatifs à un fichier de données extraits d'un manifeste de fichiers sont mis en forme sur plusieurs lignes par souci de lisibilité.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

Fichiers de données

DynamoDB peut exporter les données de votre table dans deux formats : DynamoDB JSON et Amazon Ion. Quel que soit le format que vous choisissez, vos données sont écrites dans plusieurs fichiers compressés nommés par les clés. Ces fichiers sont également répertoriés dans le fichier `manifest-files.json`.

La structure du répertoire de votre compartiment Amazon S3 après une exportation complète contient l'ensemble de vos fichiers manifeste et fichiers de données sous le dossier d'ID de l'exportation.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

JSON DynamoDB

Une exportation de table au format JSON DynamoDB comprend plusieurs objets `Item`. Chaque objet est au format JSON regroupé standard de DynamoDB.

Lorsque des analyseurs personnalisés sont créés pour les données d'exportation JSON DynamoDB, le format est JSON Lines. Cela signifie que des sauts de ligne sont utilisés en tant que délimiteurs d'éléments. Bon nombre de services AWS dont Athena et AWS Glue, analysent automatiquement ce format.

Dans l'exemple suivant, un élément d'une exportation JSON DynamoDB a été mis en forme sur plusieurs lignes par souci de lisibilité.

```
{
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

Amazon Ion

Amazon Ion est un format de sérialisation de données hiérarchique riche, auto-descriptif et conçu pour répondre aux défis de développement rapide, de découplage et d'efficacité rencontrés lors

de la conception d'architectures orientées service à grande échelle. DynamoDB prend en charge l'exportation de données de table au format texte d'Ion, qui est un sur-ensemble de JSON.

Lorsque vous exportez une table au format Ion, les types de données DynamoDB utilisés dans la table sont mappés à des types de données Ion. Les ensembles DynamoDB utilisent des annotations de type Ion pour clarifier type de données utilisé dans la table source.

Le tableau suivant répertorie le mappage des types de données DynamoDB aux types de données Ion :

Type de données DynamoDB	Représentation Ion
String (S)	string
Boolean (BOOL)	bool
Number (N)	décimal
Binary (B)	blob
Set (SS, NS, BS)	list (avec annotation de type $DynamoDB_SS, $DynamoDB_ns ou $DynamoDB_bs)
Liste	list
Map	struct

Les éléments dans une exportation Ion sont délimités par des sauts de ligne. Chaque ligne commence par un marqueur de version Ion, suivi d'un élément au format Ion. Dans l'exemple suivant, un élément d'une exportation Ion a été mis en forme sur plusieurs lignes par souci de lisibilité.

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
```

```
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
}
```

Sortie d'une exportation incrémentielle

Fichiers manifestes

DynamoDB crée des fichiers manifeste, ainsi que leurs fichiers de somme de contrôle, dans le compartiment S3 spécifié pour chaque demande d'exportation.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Vous choisissez un **`export-prefix`** lorsque vous demandez une exportation de table. Cela vous aide à organiser les fichiers dans le compartiment S3 de destination. L'**`ExportId`** est un jeton unique généré par le service qui vise à éviter que plusieurs exportations vers un même compartiment S3 et `export-prefix` ne se remplacent les unes les autres.

L'exportation crée au moins un fichier par partition. Pour les partitions vides, votre demande d'exportation crée un fichier vide. Tous les éléments de chaque fichier proviennent de l'espace de clés haché de cette partition particulière.

> **Note**
>
> DynamoDB crée également un fichier vide nommé `_started` dans le même répertoire que les fichiers manifestes. Ce fichier vérifie que le compartiment de destination est accessible en écriture et que l'exportation a commencé. Il peut être supprimé en toute sécurité.

Manifeste de récapitulatif

Le fichier `manifest-summary.json` contient des informations récapitulatives sur la tâche d'exportation. Cela vous permet de savoir quels fichiers de données du dossier de données partagé sont associés à cette exportation. Son format est le suivant :

```
{
```

```
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
 "startTime": "2023-09-19T04:20:18.000Z",
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```


Manifeste de fichiers

Le fichier `manifest-files.json` contient des informations sur les fichiers contenant les données de votre table exportée. Le fichier étant au format JSON Lines, de nouvelles lignes sont utilisées comme délimiteurs d'éléments. Dans l'exemple suivant, les détails relatifs à un fichier de données extraits d'un manifeste de fichiers sont mis en forme sur plusieurs lignes par souci de lisibilité.


```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```


Fichiers de données

DynamoDB peut exporter les données de votre table dans deux formats : DynamoDB JSON et Amazon Ion. Quel que soit le format que vous choisissez, vos données sont écrites dans plusieurs fichiers compressés nommés par les clés. Ces fichiers sont également répertoriés dans le fichier `manifest-files.json`.

Les fichiers de données dans le cas des exportations incrémentielles sont tous contenus dans un dossier de données commun au sein de votre compartiment S3. Vos fichiers manifeste se trouvent sous le dossier d'ID de votre exportation.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

Dans les fichiers de votre exportation, la sortie de chaque élément comprend un horodatage qui représente le moment auquel cet élément a été mis à jour dans votre table et une structure de données qui indique s'il s'agissait d'une opération `insert`, `update` ou `delete`. L'horodatage est basé sur une horloge système interne et peut varier par rapport à l'horloge de votre application. Pour les exportations incrémentielles, vous pouvez choisir entre deux types de vue d'exportation pour votre structure de sortie : ancienne et nouvelle images ou nouvelles images uniquement.

- Nouvelle image indique l'état le plus récent de l'élément
- Ancienne image indique l'état de l'élément juste avant la date et l'heure de début spécifiées

Les types de vue peuvent être utiles si vous souhaitez voir comment l'élément a changé durant la période d'exportation. Cela peut aussi s'avérer utile pour mettre à jour efficacement vos systèmes en aval, en particulier si leur clé de partition est différente de votre clé de partition DynamoDB.

Vous pouvez déterminer si un élément de la sortie de votre exportation incrémentielle était une opération `insert`, `update` ou `delete` en examinant la structure de la sortie. La structure de

l'exportation incrémentielle et ses opérations correspondantes sont résumées dans le tableau ci-dessous pour les deux types de vue d'exportation.

Opération	Nouvelles images uniquement	Ancienne et nouvelle images
Insert	Clés + nouvelle image	Clés + nouvelle image
Mettre à jour	Clés + nouvelle image	Clés + nouvelle image + ancienne image
Suppression	Clés	Clés + ancienne image
Insert + delete	Pas de sortie	Pas de sortie

JSON DynamoDB

Une exportation de table au format JSON de DynamoDB comprend un horodatage des métadonnées qui indique l'heure d'écriture de l'élément, suivi des clés de l'élément et des valeurs. Voici un exemple de sortie JSON DynamoDB utilisant le type de vue d'exportation Ancienne et nouvelle images.

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
     "LastName": {
```

```
        "S": "Don"
      }
    }
}

// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#200"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Grace"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Smith"
     }
   }
}

// Ex 3: Delete
```

```
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#300"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#300"
     },
     "FirstName": {
       "S": "Jose"
     },
     "LastName": {
       "S": "Hernandez"
     }
   }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

Amazon Ion

Amazon Ion est un format de sérialisation de données hiérarchique riche, auto-descriptif et conçu pour répondre aux défis de développement rapide, de découplage et d'efficacité rencontrés lors de la conception d'architectures orientées service à grande échelle. DynamoDB prend en charge l'exportation de données de table au format texte d'Ion, qui est un sur-ensemble de JSON.

Lorsque vous exportez une table au format Ion, les types de données DynamoDB utilisés dans la table sont mappés à des types de données Ion. Les ensembles DynamoDB utilisent des annotations de type Ion pour clarifier type de données utilisé dans la table source.

Le tableau suivant répertorie le mappage des types de données DynamoDB aux types de données Ion :

Type de données DynamoDB	Représentation Ion
String (S)	string
Boolean (BOOL)	bool
Number (N)	décimal
Binary (B)	blob
Set (SS, NS, BS)	list (avec annotation de type $DynamoDB_SS, $DynamoDB_ns ou $DynamoDB_bs)
Liste	list
Map	struct

Les éléments dans une exportation Ion sont délimités par des sauts de ligne. Chaque ligne commence par un marqueur de version Ion, suivi d'un élément au format Ion. Dans l'exemple suivant, un élément d'une exportation Ion a été mis en forme sur plusieurs lignes par souci de lisibilité.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
```

```
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

Intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse

L'intégration zéro ETL DynamoDB à Amazon SageMaker Lakehouse élimine le besoin de créer des pipelines de transfert de données personnalisés en répliquant automatiquement les données DynamoDB vers Amazon SageMaker Lakehouse. Cette intégration sans programmation permet aux clients d'exécuter des charges de travail analytiques sur leurs données DynamoDB à l'aide d'Amazon SageMaker Lakehouse sans consommer la capacité des tables DynamoDB. L'intégration exporte automatiquement les données de votre table et actualise la cible, généralement dans un délai de 15 à 30 minutes.

Rubriques

- Intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse

Intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse

La configuration d'une intégration entre la table DynamoDB et Amazon SageMaker Lakehouse nécessite certaines conditions préalables, telles que la configuration des rôles IAM utilisés par AWS Glue pour accéder aux données depuis la source et écrire sur la cible, et l'utilisation de clés KMS pour chiffrer les données à l'emplacement intermédiaire ou cible.

Rubriques

- Conditions préalables à la création d'une intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse
- Création d'une intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse
- Affichage des métriques CloudWatch à des fins d'intégration

Conditions préalables à la création d'une intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse

Pour configurer une intégration zéro ETL avec une source DynamoDB, vous devez configurer une politique d'accès basée sur les ressources (RBAC) permettant à AWS Glue d'accéder aux données de la table DynamoDB et de les exporter. Cette politique doit inclure des autorisations spécifiques telles que `ExportTableToPointInTime`, `DescribeTable` et `DescribeExport` avec des conditions restreignant l'accès à un Compte AWS et à une région spécifiques. Consultez Configuration d'une source Amazon DynamoDB pour plus d'informations.

La reprise ponctuelle (PITR) doit être activée pour la table, et vous pouvez appliquer la politique à l'aide de commandes AWS CLI. La politique peut être affinée en spécifiant l'ARN d'intégration complet pour un contrôle d'accès plus restrictif. Pour plus d'informations, consultez Conditions préalables à la configuration d'une intégration zéro ETL.

Création d'une intégration zéro ETL de DynamoDB à Amazon SageMaker Lakehouse

Après avoir rempli les conditions préalables à l'intégration, vous pouvez créer, modifier ou supprimer l'intégration zéro ETL en suivant les instructions ci-dessous :

Création d'une intégration

Pour créer une intégration

1. Connectez-vous à la Console de gestion AWS et ouvrez la console Amazon DynamoDB à l'adresse https://console.aws.amazon.com/dynamodbv2.
2. Dans le volet de navigation, choisissez Intégrations.
3. Sélectionnez Créez une intégration zéro ETL avec Amazon SageMaker Lakehouse, puis choisissez Suivant.
4. Pour créer une intégration, consultez Création d'une intégration.
5. Pour modifier une intégration, consultez Modification d'une intégration.
6. Pour supprimer une intégration, consultez Suppression d'une intégration.
7. Pour configurer une intégration entre comptes, consultez Configuration d'une intégration entre comptes.

Activation du compactage sur les tableaux Amazon S3 cibles

Vous pouvez activer le compactage pour améliorer les performances des requêtes dans Amazon Athena.

Commencez par effectuer la configuration préalable pour les ressources de compactage, y compris la configuration du rôle IAM nécessaire. Reportez-vous à la documentation de Lake Formation pour connaître les étapes détaillées de configuration des rôles IAM. Consultez Optimizing tables for compaction.

Pour activer le compactage sur la table AWS Glue créée lors de l'intégration, suivez le processus d'activation du compactage de Lake Formation. Cela vous permettra d'optimiser les performances de votre table et l'efficacité des requêtes.

Affichage des métriques CloudWatch à des fins d'intégration

Une fois l'intégration terminée, vous pouvez voir ces métriques CloudWatch et les notifications EventBridge générées dans votre compte pour chaque tâche AWS Glue. Pour plus d'informations, consultez Surveillance d'une intégration.

Intégration zéro ETL de DynamoDB à Amazon OpenSearch Service

Amazon DynamoDB propose une intégration zéro ETL à Amazon OpenSearch Service au moyen du Plug-in DynamoDB pour OpenSearch Ingestion. Amazon OpenSearch Ingestion offre une expérience entièrement gérée et sans programmation pour l'ingestion de données dans Amazon OpenSearch Service.

Avec le plug-in DynamoDB pour OpenSearch Ingestion, vous pouvez utiliser une ou plusieurs tables DynamoDB comme source d'ingestion vers un ou plusieurs index OpenSearch Service. Vous pouvez parcourir et configurer vos pipelines OpenSearch Ingestion avec DynamoDB comme source à partir d'OpenSearch Ingestion ou de DynamoDB Integrations dans la AWS Management Console.

- Commencez avec OpenSearch Ingestion en vous reportant aux rubriques appropriées dans OpenSearch Ingestion getting started guide.
- Découvrez les prérequis et toutes les options de configuration du plug-in DynamoDB dans la documentation du plugin DynamoDB pour OpenSearch Ingestion.

Fonctionnement

Le plug-in utilise l'exportation DynamoDB vers Amazon S3 pour créer un instantané initial à charger dans OpenSearch. Une fois l'instantané chargé, le plug-in utilise les flux DynamoDB pour répliquer toute modification ultérieure en temps quasi réel. Chaque élément est traité comme un événement dans OpenSearch Ingestion et peut être modifié avec les plug-ins du processeur. Vous pouvez supprimer des attributs ou créer des attributs composites et les envoyer à différents index via des itinéraires.

La reprise ponctuelle (PITR) doit être activée pour l'utilisation de l'exportation vers Amazon S3. DynamoDB Streams doit également être activé (avec l'option Nouvelles et anciennes images sélectionnée) pour pouvoir l'utiliser. Il est possible de créer un pipeline sans prendre de capture instantanée en excluant les paramètres d'exportation.

Vous pouvez également créer un pipeline contenant uniquement un instantané et aucune mise à jour en excluant les paramètres des flux. Le plug-in n'utilise pas le débit de lecture ou d'écriture sur votre table, il est donc sûr à utiliser sans affecter votre trafic de production. Il existe des limites au nombre d'utilisateurs parallèles sur un flux. Vous devez en tenir compte avant de créer cette intégration ou d'autres intégrations. Pour d'autres considérations, consultez the section called "Bonnes pratiques d'intégration".

Pour les pipelines simples, une seule unité de calcul OpenSearch (OCU) peut traiter environ 1 Mo d'écriture par seconde. C'est l'équivalent d'environ 1 000 unités de demande d'écriture (WCU). En fonction de la complexité de votre pipeline et d'autres facteurs, vous pouvez obtenir plus ou moins que cela.

OpenSearch Ingestion prend en charge une file d'attente de lettres mortes (DLQ) pour les événements à l'origine d'erreurs irrécupérables. En outre, le pipeline peut reprendre là où il s'était arrêté sans intervention de l'utilisateur, même en cas d'interruption de service avec DynamoDB, le pipeline ou Amazon OpenSearch Service.

Si l'interruption dure plus de 24 heures, cela peut entraîner la perte des mises à jour. Toutefois, le pipeline peut continuer à traiter les mises à jour encore disponibles une fois la disponibilité rétablie. Vous devrez créer un nouvel index pour corriger les irrégularités dues aux événements supprimés, sauf s'ils se trouvent dans la file d'attente de lettres mortes.

Pour tous les paramètres et les détails du plug-in, consultez la documentation du plug-in OpenSearch Ingestion DynamoDB.

Expérience de création intégrée via la console

DynamoDB et OpenSearch Service intègrent une expérience intégrée dans la AWS Management Console, ce qui rationalise le processus de mise en route. Lorsque vous suivez ces étapes, le service sélectionne automatiquement le plan DynamoDB et ajoute les informations DynamoDB appropriées pour vous.

Pour créer une intégration, consultez les rubriques appropriées dans OpenSearch Ingestion getting started guide. Lorsque vous arrivez à Step 3 : Create a pipeline, remplacez les étapes 1 et 2 par les étapes suivantes :

1. Accédez à la console DynamoDB.
2. Dans le panneau de navigation de gauche, choisissez Integration.
3. Sélectionnez la table DynamoDB que vous souhaitez répliquer dans OpenSearch.
4. Choisissez Create.

À partir de là, vous pouvez continuer avec le reste du didacticiel.

Étapes suivantes

Pour mieux comprendre comment DynamoDB s'intègre à OpenSearch Service, consultez ce qui suit :

- Mise en route avec Amazon OpenSearch Ingestion
- Configuration et exigences du plug-in DynamoDB

Gestion des modifications avec rupture apportées à votre index

OpenSearch peut ajouter dynamiquement de nouveaux attributs à votre index. Cependant, une fois que votre modèle de mappage a été défini pour une clé donnée, vous devez prendre des mesures supplémentaires pour le modifier. En outre, si votre modification vous oblige à retraiter toutes les données de votre table DynamoDB, vous devrez prendre des mesures pour lancer une nouvelle exportation.

> **Note**
>
> Dans toutes ces options, vous pouvez toujours rencontrer des problèmes si votre table DynamoDB présente des conflits de type avec le modèle de mappage que vous avez spécifié. Assurez-vous qu'une file d'attente de lettres mortes (DLQ) est activée (même

en cours de développement). Cela permet de mieux comprendre ce qui ne va pas avec l'enregistrement à l'origine d'un conflit lorsqu'il est indexé dans votre index sur OpenSearch.

Rubriques

- Fonctionnement
- Supprimez votre index et réinitialisez le pipeline (option centrée sur le pipeline)
- Recréez votre index et réinitialisez le pipeline (option centrée sur l'index)
- Création d'un nouvel index et d'un nouveau récepteur (option en ligne)
- Bonnes pratiques pour éviter et déboguer les conflits de type

Fonctionnement

Voici un bref aperçu des mesures prises lors de la gestion des modifications de rupture apportées à votre index. Consultez les procédures détaillées dans les sections qui suivent.

- Stop and start the pipeline : cette option réinitialise l'état du pipeline, qui redémarrera avec une nouvelle exportation complète. Il n'est pas destructif et ne supprime donc pas votre index ni aucune donnée dans DynamoDB. Si vous ne créez pas un nouvel index avant cela, vous risquez de rencontrer un grand nombre d'erreurs dues à des conflits de versions, car l'exportation tente d'insérer des documents plus anciens que le document `_version` actuel dans l'index. Vous pouvez ignorer ces erreurs en toute sécurité. Le pipeline ne vous sera pas facturé pendant qu'il est arrêté.
- Update the pipeline : cette option met à jour la configuration du pipeline selon une approche bleu/vert, sans perdre d'état. Si vous apportez des modifications importantes à votre pipeline (par exemple en ajoutant de nouveaux itinéraires, de nouveaux index ou de nouvelles clés à des index existants), vous devrez peut-être effectuer une réinitialisation complète du pipeline et recréer votre index. Cette option ne permet pas d'effectuer une exportation complète.
- Delete and recreate the index : cette option supprime vos données et les paramètres de mappage de votre index. Vous devez effectuer cette opération avant d'apporter des modifications de rupture à vos mappages. Cela interrompra toutes les applications qui s'appuient sur l'index jusqu'à ce que celui-ci soit recréé et synchronisé. La suppression de l'index ne lance pas une nouvelle exportation. Vous ne devez supprimer votre index qu'après avoir mis à jour votre pipeline. Dans le cas contraire, il se peut que votre index soit recréé avant que vous ne mettiez à jour vos paramètres.

Supprimez votre index et réinitialisez le pipeline (option centrée sur le pipeline)

Cette méthode est souvent l'option la plus rapide si vous êtes encore en phase de développement. Vous allez supprimer votre index dans OpenSearch Service, puis arrêter et démarrer votre pipeline pour lancer une nouvelle exportation de toutes vos données. Cela garantit qu'aucun modèle de mappage n'entre en conflit avec les index existants et qu'aucune perte de données n'est due à une table traitée incomplète.

1. Arrêtez le pipeline par l'intermédiaire de la AWS Management Console ou à l'aide de l'opération d'API StopPipeline avec l'AWS CLI ou un kit SDK.
2. Mettez à jour la configuration de votre pipeline avec vos nouvelles modifications.
3. Supprimez votre index dans OpenSearch Service, soit via un appel d'API `REST`, soit via votre tableau de bord OpenSearch.
4. Démarrez le pipeline via la console ou en utilisant l'opération d'API `StartPipeline` avec l'AWS CLI ou un kit SDK.

   > **Note**
   >
   > Cela initie une nouvelle exportation complète, qui entraînera des coûts supplémentaires.

5. Surveillez tout problème inattendu, car une nouvelle exportation est générée pour créer le nouvel index.
6. Vérifiez que l'index correspond à vos attentes dans OpenSearch Service.

Une fois l'exportation terminée et la lecture à partir du flux reprise, les données de votre table DynamoDB seront désormais disponibles dans l'index.

Recréez votre index et réinitialisez le pipeline (option centrée sur l'index)

Cette méthode fonctionne bien si vous devez effectuer de nombreuses itérations sur la conception de l'index dans OpenSearch Service avant de reprendre le pipeline depuis DynamoDB. Cela peut être utile pour le développement lorsque vous souhaitez itérer très rapidement sur vos modèles de recherche et éviter d'attendre que de nouvelles exportations soient effectuées entre chaque itération.

1. Arrêtez le pipeline par l'intermédiaire de la AWS Management Console ou en appelant l'opération d'API StopPipeline à l'aide de l'AWS CLI ou d'un kit SDK.

2. Supprimez et recréez votre index dans OpenSearch avec le modèle de mappage que vous souhaitez utiliser. Vous pouvez insérer manuellement des exemples de données pour confirmer que vos recherches fonctionnent comme prévu. Si vos exemples de données sont susceptibles d'entrer en conflit avec des données provenant de DynamoDB, veillez à les supprimer avant de passer à l'étape suivante.
3. Si vous avez un modèle d'indexation dans votre pipeline, supprimez-le ou remplacez-le par celui que vous avez déjà créé dans OpenSearch Service. Assurez-vous que le nom de votre index correspond au nom indiqué dans le pipeline.
4. Démarrez le pipeline à partir de la console ou en appelant l'opération d'API `StartPipeline` avec l'AWS CLI ou un kit SDK.

   > **Note**
   >
   > Cela initie une nouvelle exportation complète, qui entraînera des coûts supplémentaires.

5. Surveillez tout problème inattendu, car une nouvelle exportation est générée pour créer le nouvel index.

Une fois l'exportation terminée et la lecture à partir du flux reprise, les données de votre table DynamoDB seront désormais disponibles dans l'index.

Création d'un nouvel index et d'un nouveau récepteur (option en ligne)

Cette méthode fonctionne bien si vous devez mettre à jour votre modèle de mappage alors que vous utilisez actuellement votre index en production. Cela crée un tout nouvel index, vers lequel vous devrez déplacer votre application une fois celle-ci synchronisée et validée.

> **Note**
>
> Cela créera un autre utilisateur sur le flux. Cela peut être un problème si vous avez également d'autres utilisateurs comme AWS Lambda ou des tables globales. Vous devrez peut-être suspendre les mises à jour de votre pipeline existant pour créer la capacité de charger le nouvel index.

1. Créez un nouveau pipeline avec de nouveaux paramètres et un nom d'index différent.
2. Surveillez le nouvel index pour détecter tout problème imprévu.

3. Basculez l'application vers le nouvel index.
4. Arrêtez et supprimez l'ancien pipeline après avoir vérifié que tout fonctionne correctement.

Bonnes pratiques pour éviter et déboguer les conflits de type

- Utilisez toujours une file d'attente de lettres mortes (DLQ) pour faciliter le débogage en cas de conflits de types.
- Utilisez toujours un modèle d'index avec des mappages et définissez `include_keys`. Bien qu'OpenSearch Service mappe de manière dynamique les nouvelles clés, cela peut entraîner des problèmes liés à des comportements inattendus (comme s'attendre à ce que quelque chose soit un `GeoPoint`, alors que celui-ci est créé sous la forme `string` ou `object`) ou à des erreurs (comme le fait d'avoir un `number` qui soit un mélange des valeurs `long` et `float`).
- Si vous devez continuer à utiliser votre index existant en production, vous pouvez également remplacer l'une des étapes de suppression d'index précédentes en renommant simplement votre index dans le fichier de configuration de votre pipeline. Cela crée un tout nouvel index. Votre application devra ensuite être mise à jour pour pointer vers le nouvel index une fois qu'elle sera terminée.
- Si vous rencontrez un problème de conversion de type que vous résolvez avec un processeur, vous pouvez le tester avec `UpdatePipeline`. Pour ce faire, vous devez arrêter et démarrer ou traiter vos files d'attente de lettres mortes afin de corriger les documents précédemment ignorés qui contenaient des erreurs.

Bonnes pratiques d'utilisation de l'intégration zéro ETL à DynamoDB et OpenSearch Service

DynamoDB comporte une intégration zéro ETL DynamoDB à Amazon OpenSearch Service. Pour plus d'informations, consultez les rubriques DynamoDB plugin for OpenSearch Ingestion et bonnes pratiques spécifiques relatives à Amazon OpenSearch Service.

Configuration

- Indexez uniquement les données sur lesquelles vous devez effectuer des recherches. Utilisez toujours un modèle de mappage (`template_type: index_template` et `template_content`) et `include_keys` pour l'implémenter.

- Surveillez vos journaux pour détecter les erreurs liées à des conflits de types. OpenSearch Service s'attend à ce que toutes les valeurs d'une clé donnée soient du même type. Il génère des exceptions en cas de non-concordance. Si vous rencontrez l'une de ces erreurs, vous pouvez ajouter un processeur pour détecter qu'une clé donnée a toujours la même valeur.
- Utilisez généralement la valeur `primary_key` des métadonnées pour la valeur `document_id`. Dans OpenSearch Service, l'ID du document est l'équivalent de la clé primaire dans DynamoDB. L'utilisation de la clé primaire facilitera la recherche de votre document et garantira que les mises à jour y sont systématiquement répliquées sans conflits.

  Vous pouvez utiliser la fonction d'annotations `getMetadata` pour obtenir votre clé primaire (par exemple, `document_id: "${getMetadata('primary_key')}"`). Si vous utilisez une clé primaire composite, la fonction d'annotations les concaténera pour vous.
- Utilisez généralement la valeur `opensearch_action` des métadonnées pour le paramètre `action`. Cela garantira que les mises à jour sont répliquées de telle sorte que les données d'OpenSearch Service correspondent à l'état le plus récent de DynamoDB.

  Vous pouvez utiliser la fonction d'annotations `getMetadata` pour obtenir votre clé primaire (par exemple, `action: "${getMetadata('opensearch_action')}"`). Vous pouvez également obtenir le type d'événement de diffusion `dynamodb_event_name` pour des cas d'utilisation tels que le filtrage. Cependant, vous ne devez généralement pas l'utiliser pour le paramètre `action`.

Observabilité

- Utilisez toujours une file d'attente de lettres mortes (DLQ) sur vos récepteurs OpenSearch pour gérer les événements supprimés. DynamoDB est généralement moins structuré qu'OpenSearch Service, et il est toujours possible qu'un imprévu se produise. Une file d'attente de lettres mortes vous permet de récupérer des événements individuels et même d'automatiser le processus de récupération. Cela vous évitera d'avoir à reconstruire l'intégralité de votre index.
- Définissez toujours des alertes indiquant que le délai de réplication ne dépasse pas le montant prévu. Il est généralement prudent de prévoir une minute sans que l'alerte soit trop bruyante. Cela peut varier en fonction de l'importance de votre trafic d'écriture et des paramètres de votre unité de calcul OpenSearch (OCU) sur le pipeline.

  Si le délai de réplication dépasse 24 heures, votre flux commencera à supprimer des événements et vous rencontrerez des problèmes de précision, à moins que vous ne reconstruisiez complètement votre index à partir de zéro.

Mise à l’échelle

- Utilisez l’autoscaling pour les pipelines afin d’augmenter verticalement ou de réduire les unités OCU afin de mieux les adapter à la charge de travail.
- Pour les tables de débit provisionnées sans autoscaling, nous vous recommandons de définir les unités OCU en fonction de vos unités de capacité d’écriture (WCU) divisées par 1 000. Définissez le minimum à 1 unité OCU en dessous de ce montant (mais au moins 1), et définissez le maximum à au moins 1 unité OCU au-dessus de ce montant.
  - Formule :

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - Exemple : 25 000 unités WCU sont provisionnées dans votre table. Les unités OCU de votre pipeline doivent être définies sur un minimum de 24 (25 000/1 000 - 1) et un maximum d’au moins 26 (25 000/1 000 + 1).
- Pour les tables de débit provisionnées avec autoscaling, nous vous recommandons de définir les unités OCU en fonction de vos unités de capacité d’écriture (WCU) minimum et maximum divisées par 1 000. Définissez le minimum à 1 unité OCU en dessous du minimum de DynamoDB, et définissez le maximum à au moins 1 unité OCU au-dessus du maximum de DynamoDB.
  - Formule :

    ```
    OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
    OCU_maximum = (table_maximum_WCU / 1000) + 1
    ```

  - Exemple : votre table dispose d’une politique d’autoscaling avec un minimum de 8 000 et un maximum de 14 000. Les unités OCU de votre pipeline doivent être définies sur un minimum de 7 (8 000/1 000 - 1) et un maximum d’au moins 15 (14 000/1 000 + 1).
- Pour les tables de débit à la demande, nous vous recommandons de définir les unités OCU en fonction de votre pic et de votre vallée habituels pour les unités de demande d’écriture par seconde. Il se peut que vous deviez établir une moyenne sur une période plus longue, en fonction de l’agrégation dont vous disposez. Définissez le minimum à 1 unité OCU en dessous du minimum de DynamoDB, et définissez le maximum à au moins 1 unité OCU au-dessus du maximum de DynamoDB.
  - Formule :

    ```
    # Assuming we have writes aggregated at the minute level
    ```

```
OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
```

- Exemple : votre table présente une vallée moyenne de 300 unités de demande d'écriture par seconde et un pic moyen de 4 300. Les unités OCU de votre pipeline doivent être définies sur un minimum de 1 (300/1 000 - 1, mais au moins 1) et un maximum de 5 (4 300/1 000 + 1).

- Suivez les bonnes pratiques en matière de mise à l'échelle de vos index OpenSearch Service de destination. Si vos index sont sous-dimensionnés, cela ralentira l'ingestion à partir de DynamoDB et peut entraîner des retards.

> **Note**
>
> `GREATEST` est une fonction SQL qui, à partir d'un ensemble d'arguments, renvoie l'argument ayant la plus grande valeur.

Intégration de DynamoDB à Amazon EventBridge

Amazon DynamoDB propose DynamoDB Streams pour la capture des données de modification, ce qui permet de capturer les modifications au niveau des éléments dans les tables DynamoDB. DynamoDB Streams peut invoquer des fonctions Lambda pour traiter ces modifications, ce qui permet une intégration basée sur les événements à d'autres services et applications. DynamoDB Streams prend également en charge le filtrage, ce qui permet un traitement des événements efficace et ciblé.

DynamoDB Streams prend en charge jusqu'à deux consommateurs simultanés par partition et prend en charge le filtrage via le filtrage des événements Lambda, de sorte que seuls les éléments répondant à des critères spécifiques soient traités. Certains clients peuvent avoir besoin de prendre en charge plus de deux clients. D'autres peuvent avoir besoin d'enrichir les événements de modification avant qu'ils ne soient traités, ou d'utiliser un filtrage et un routage plus avancés.

L'intégration de DynamoDB à EventBridge peut répondre à ces exigences.

Amazon EventBridge est un service sans serveur qui utilise des événements pour connecter les composants de l'application entre eux, ce qui vous permet de créer plus facilement des applications évolutives axées sur des événements. EventBridge propose une intégration native à Amazon DynamoDB via EventBridge Pipes, permettant ainsi un flux de données transparent entre DynamoDB

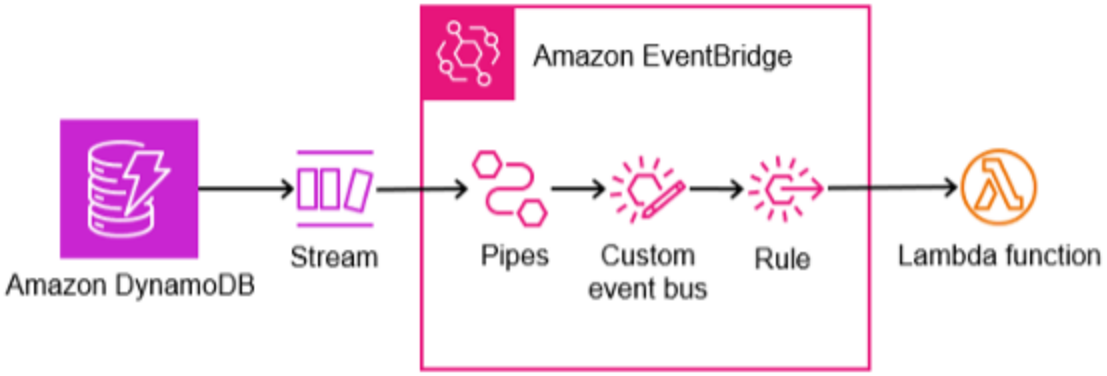
et un bus EventBridge. Ce bus peut ensuite être diffusé vers plusieurs applications et services via un ensemble de règles et de cibles.

Rubriques

- Fonctionnement
- Mise à jour d'une intégration via la console
- Étapes suivantes

Fonctionnement

L'intégration entre les canaux DynamoDB et EventBridge utilise DynamoDB Streams pour capturer une séquence chronologique des modifications au niveau des éléments dans une table DynamoDB. Chaque enregistrement capturé de cette manière contient les données modifiées dans la table.



Un canal EventBridge utilise les événements provenant de DynamoDB Streams et les achemine vers une cible telle qu'un bus EventBridge (un bus d'événements est un routeur qui reçoit des événements et les transmet à des destinations, également appelées cibles). La livraison dépend des règles qui correspondent au contenu de l'événement. Le canal offre également la possibilité d'effectuer un filtrage sur des événements spécifiques et d'enrichir leurs données d'événement avant de les envoyer à la cible.

Bien qu'EventBridge prenne en charge plusieurs types de cibles, il est courant d'utiliser une fonction Lambda comme cible lors de la mise en œuvre d'une conception en éventail. L'exemple suivant démontre une intégration à une fonction Lambda cible.

Mise à jour d'une intégration via la console

Suivez les étapes ci-dessous pour créer une intégration via la AWS Management Console.

1. Activez DynamoDB Streams sur la table source en suivant les étapes décrites dans la section Activation d'un flux du guide du développeur DynamoDB. Si DynamoDB Streams est déjà activé dans la table source, vérifiez qu'il y a actuellement moins de deux consommateurs. Les consommateurs peuvent être des fonctions Lambda, des tables globales DynamoDB, des intégrations zéro ETL d'Amazon DynamoDB à Amazon OpenSearch Service ou des applications qui lisent directement à partir de flux, par exemple via l'adaptateur DynamoDB Streams Kinesis.
2. Créez un bus d'événements EventBridge en suivant les étapes décrites dans la section Création d'un bus d'événements Amazon EventBridge du Guide de l'utilisateur d'EventBridge.
   a. Lors de la création du bus d'événements, activez la découverte de schémas.
3. Créez un canal EventBridge en suivant les étapes décrites dans la section Création d'un canal Amazon EventBridge du Guide de l'utilisateur d'EventBridge.
   a. Lors de la configuration de la source, dans le champ Source, sélectionnez DynamoDB et dans le champ DynamoDB Streams, sélectionnez le nom du flux de table source.
   b. Lors de la configuration de la cible, dans le champ Service cible, sélectionnez le Bus d'événements EventBridge et dans le champ Bus d'événements comme cible, sélectionnez le bus d'événements créé à l'étape 2.
4. Écrivez un exemple d'élément dans la table DynamoDB source pour déclencher un événement. Cela permettra à EventBridge de déduire le schéma à partir de l'exemple d'élément. Ce schéma peut être utilisé pour créer des règles pour les événements de routage. Par exemple, si vous implémentez un modèle de conception qui implique de surcharger les attributs, vous souhaiterez peut-être déclencher des règles différentes en fonction de la valeur de votre clé de tri. Vous trouverez des informations sur la manière d'écrire un élément dans DynamoDB dans la section Working with items and attributes du Guide du développeur DynamoDB.
5. Créez un exemple de fonction Lambda Python à utiliser comme cible en suivant les étapes décrites dans la section Building Lambda functions with Python du Guide du développeur Lambda. Lors de la création de votre fonction, vous pouvez utiliser l'exemple de code ci-dessous pour démontrer l'intégration. Lorsqu'elle est invoquée, les éléments `NewImage` et `OldImage` reçus avec l'événement, qui peut être consulté dans CloudWatch Logs, sont imprimés.

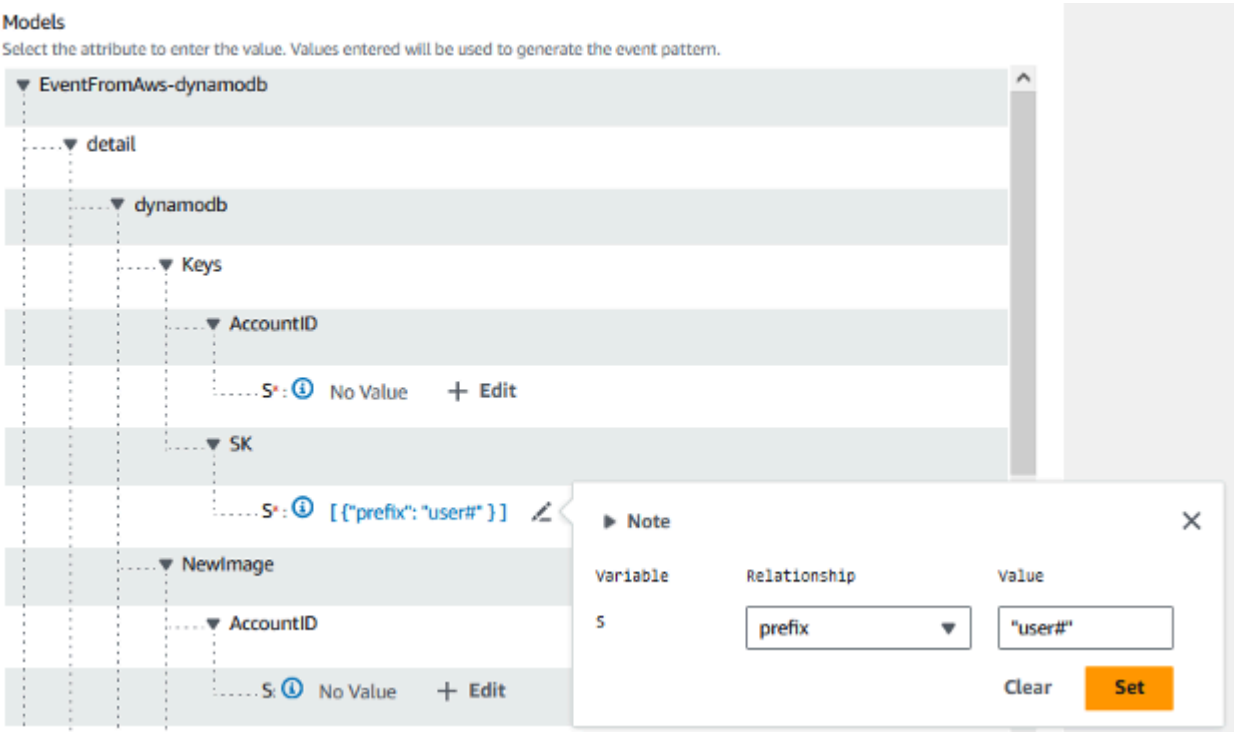
```
import json

def lambda_handler(event, context):
```

```
    dynamodb = event.get('detail', {}).get('dynamodb', {})
    new_image = dynamodb.get('NewImage')
    old_image = dynamodb.get('OldImage')

    if new_image:
        print("NewImage:", json.dumps(new_image, indent=2))
    if old_image:
        print("OldImage:", json.dumps(old_image, indent=2))

    return {'statusCode': 200, 'body': json.dumps(event)}
```

6. Créez une règle EventBridge qui acheminera les événements vers votre nouvelle fonction Lambda en suivant les étapes décrites dans la section Création d'une règle qui réagit aux événements du Guide de l'utilisateur EventBridge.

   a. Lorsque vous définissez le détail de la règle, sélectionnez le nom du bus d'événements que vous avez créé à l'étape 2 en tant que bus d'événements.

   b. Lorsque vous créez le modèle d'événement, suivez le guide du schéma existant. Ici, vous pouvez sélectionner le registre de schémas découverts et le schéma découvert pour votre événement. Cela vous permet de configurer un modèle d'événement spécifique à votre cas d'utilisation qui achemine uniquement les messages correspondant à des attributs spécifiques. Par exemple, si vous souhaitez associer uniquement sur les éléments DynamoDB où le kit SK commence par "`user#`", vous devez utiliser une configuration comme celle-ci.



c. Cliquez sur Générer un modèle d'événement au format JSON une fois que vous avez fini de concevoir un modèle par rapport à votre schéma. Si vous souhaitez plutôt associer tous les événements qui apparaissent sur DynamoDB Streams, utilisez le JSON suivant pour le modèle d'événement.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Lorsque vous sélectionnez des cibles, suivez le guide du service AWS. Pour Sélectionner une cible, choisissez « Fonction Lambda ». Dans le champ Fonction, sélectionnez la fonction Lambda que vous avez créée à l'étape 5.

7. Vous pouvez désormais arrêter la découverte de schémas sur le bus d'événement en suivant les étapes de la section the Starting or stopping schema discovery on event buses guide de l'utilisateur EventBridge.
8. Écrivez un deuxième exemple d'élément dans la table DynamoDB source pour déclencher un événement. Vérifiez que l'événement a été correctement traité à chaque étape.

   a. Consultez la métrique CloudWatch PutEventsApproximateSuccessCount pour le bus d'événements en suivant la section Monitoring Amazon EventBridge Guide de l'utilisateur EventBridge.

   b. Consultez les journaux des fonctions de votre fonction Lambda en suivant la section Monitoring and troubleshooting Lambda functions du guide du développeur Lambda. Si la fonction Lambda utilise l'exemple de code fourni, les flux `NewImage` et `OldImage` provenant de DynamoDB Streams devraient être imprimés dans le groupe de journaux CloudWatch Logs.

   c. Consultez la mesure Nombre d'erreurs et taux de réussite (%) de la fonction Lambda en suivant la section Monitoring and troubleshooting Lambda functions du Guide du développeur Lambda.

Étapes suivantes

Cet exemple présente une intégration de base avec une seule fonction Lambda en tant que cible. Pour mieux comprendre les configurations plus complexes, telles que la création de règles multiples, la création de cibles multiples, l'intégration à d'autres services et l'enrichissement des événements, consultez le guide utilisateur EventBridge complet Getting started with EventBridge.

> **Note**
>
> Soyez conscient des quotas EventBridge susceptibles d'être pertinents pour votre application. Bien que la capacité de DynamoDB Streams évolue en fonction de votre table, les quotas d'EventBridge sont distincts. Les quotas courants à prendre en compte dans une application de grande taille sont les suivants : Invocations throttle limit in transactions per second et PutEvents throttle limit in transactions per second. Ces quotas spécifient le nombre d'invocations qui peuvent être envoyées aux cibles et le nombre d'événements par seconde qui peuvent être placés dans le bus.

Intégration de DynamoDB à Amazon Managed Streaming for Apache Kafka

Amazon Managed Streaming for Apache Kafka (Amazon MSK) facilite l'ingestion et le traitement des données de streaming en temps réel grâce à un service Apache Kafka entièrement géré.

Apache Kafka est un magasin de données distribué optimisé pour l'ingestion et le traitement de données de streaming en temps réel. Kafka peut traiter des flux d'enregistrements, stocker efficacement des flux d'enregistrements dans l'ordre dans lequel les enregistrements ont été générés, et publier des flux d'enregistrements et s'abonner à des flux d'enregistrements.

Grâce à ces fonctionnalités, Apache Kafka est souvent utilisé pour créer des pipelines de données de streaming en temps réel. Un pipeline de données traite et déplace les données de manière fiable d'un système à un autre et peut jouer un rôle important dans l'adoption d'une stratégie de base de données sur mesure en facilitant l'utilisation de plusieurs bases de données qui prennent chacune en charge différents cas d'utilisation.

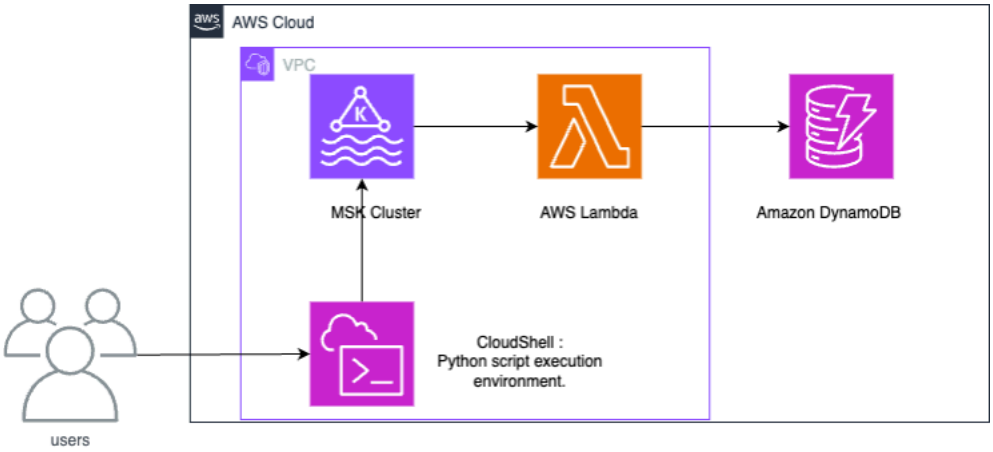
Amazon DynamoDB est une cible courante dans ces pipelines de données pour prendre en charge les applications qui utilisent des modèles de données clé-valeur ou des modèles de données de document et qui recherchent une capacité de mise à l'échelle illimitée avec des performances constantes à une milliseconde à un chiffre.

Rubriques

- Fonctionnement
- Configuration d'une intégration entre Amazon MSK et DynamoDB
- Étapes suivantes

Fonctionnement

Une intégration entre Amazon MSK et DynamoDB utilise une fonction Lambda pour utiliser les enregistrements d'Amazon MSK et les écrire dans DynamoDB.



Lambda interroge en interne les nouveaux messages de la source d'événement, puis invoque de manière synchrone la fonction Lambda cible. Les données utiles des événements de la fonction Lambda contient des lots de messages provenant d'Amazon MSK. Pour l'intégration entre Amazon MSK et DynamoDB, la fonction Lambda écrit ces messages dans DynamoDB.

Configuration d'une intégration entre Amazon MSK et DynamoDB

> **Note**
>
> Vous pouvez télécharger les ressources utilisées dans cet exemple dans le référentiel GitHub suivant.

Les étapes ci-dessous montrent comment configurer un exemple d'intégration entre Amazon MSK et Amazon DynamoDB. L'exemple représente des données générées par des appareils IoT (Internet des objets) et ingérées dans Amazon MSK. Lorsque les données sont ingérées dans Amazon MSK, elles peuvent être intégrées à des services analytiques ou à des outils tiers compatibles avec Apache Kafka, ce qui permet divers cas d'utilisation de l'analytique. L'intégration de DynamoDB permet également de rechercher des valeurs clés dans des enregistrements de d'appareils individuels.

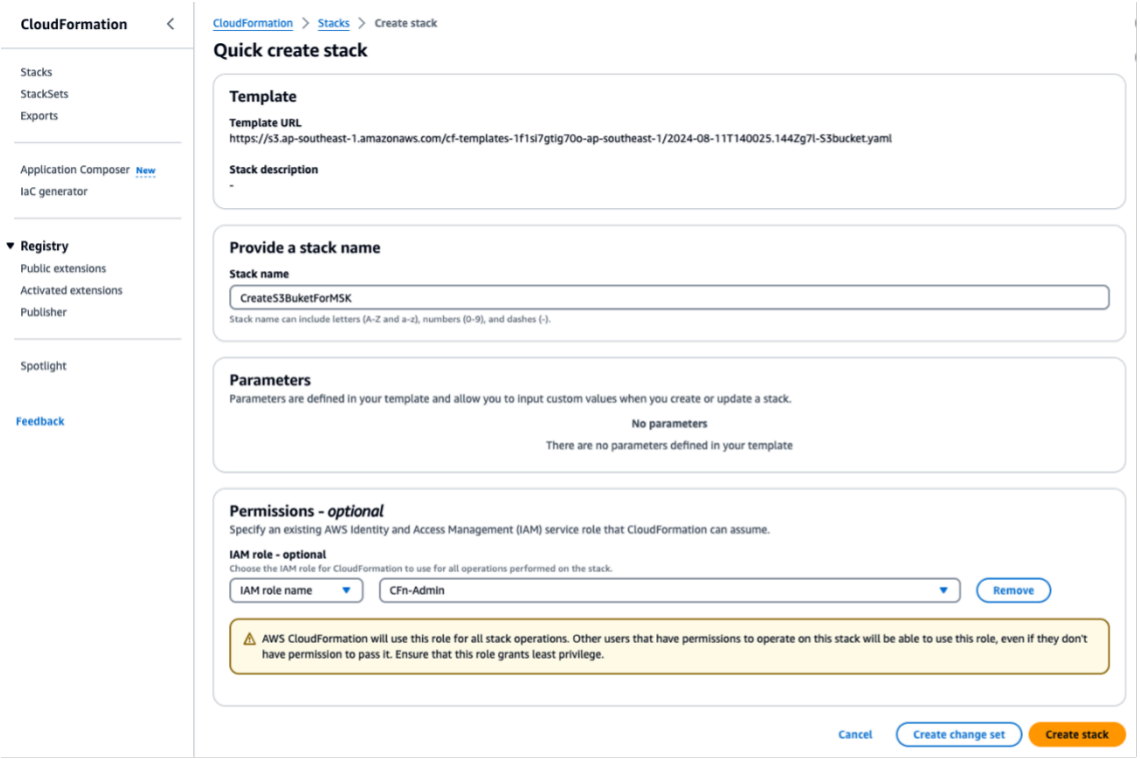
Cet exemple montre comment un script Python écrit les données des capteurs IoT sur Amazon MSK. Ensuite, une fonction Lambda écrit des éléments avec la clé de partition « `deviceid` » dans DynamoDB.

Le modèle CloudFormation fourni créera les ressources suivantes : un compartiment Amazon S3, un VPC Amazon, un cluster Amazon MSK et un AWS CloudShell pour tester les opérations de données.

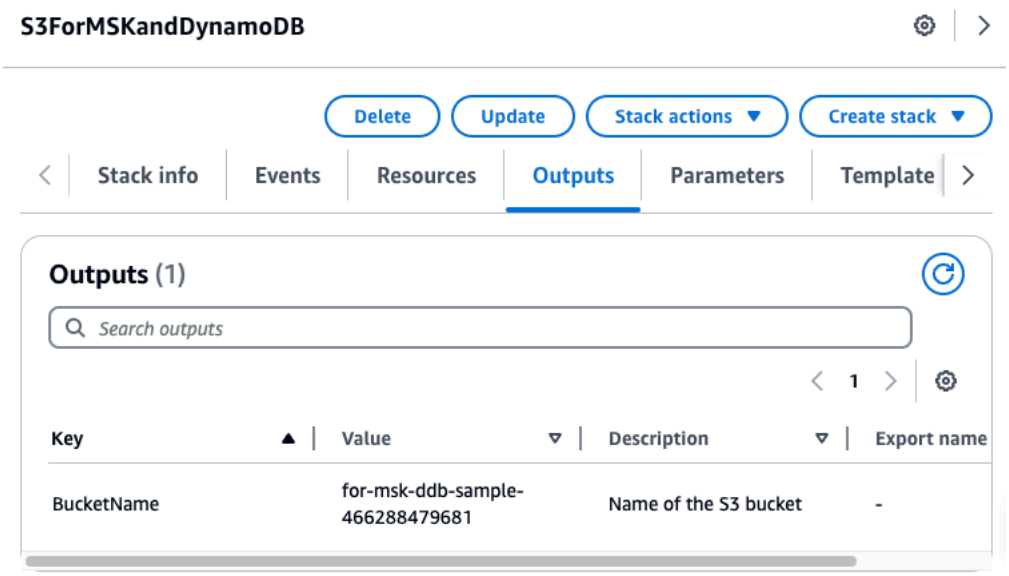
Pour générer des données de test, créez une rubrique Amazon MSK, puis créez une table DynamoDB. Vous pouvez utiliser le gestionnaire de session depuis la console de gestion pour vous connecter au système d'exploitation de CloudShell et exécuter des scripts Python.

Après avoir exécuté le modèle CloudFormation, vous pouvez terminer la création de cette architecture en effectuant les opérations suivantes.

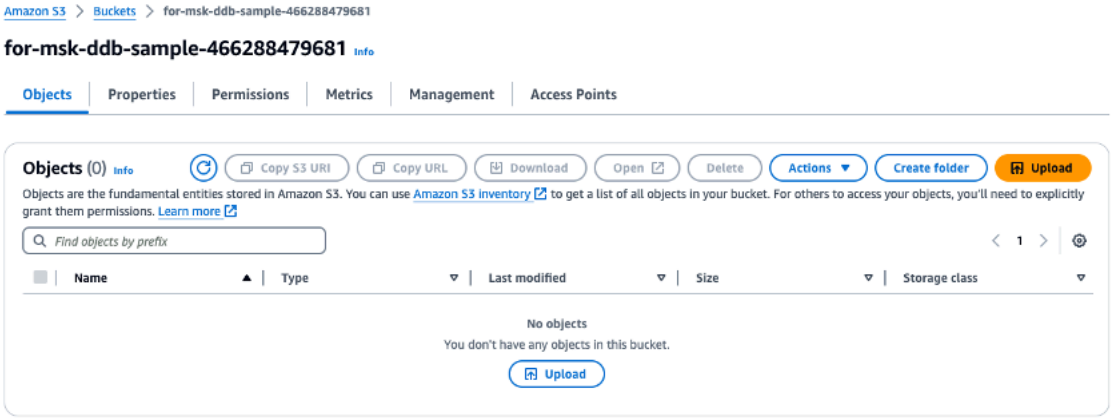
1. Exécutez le modèle CloudFormation `S3bucket.yaml` pour créer un compartiment S3. Pour tous les scripts ou opérations suivants, exécutez-les dans la même région. Entrez `ForMSKTestS3` comme nom de la pile CloudFormation.



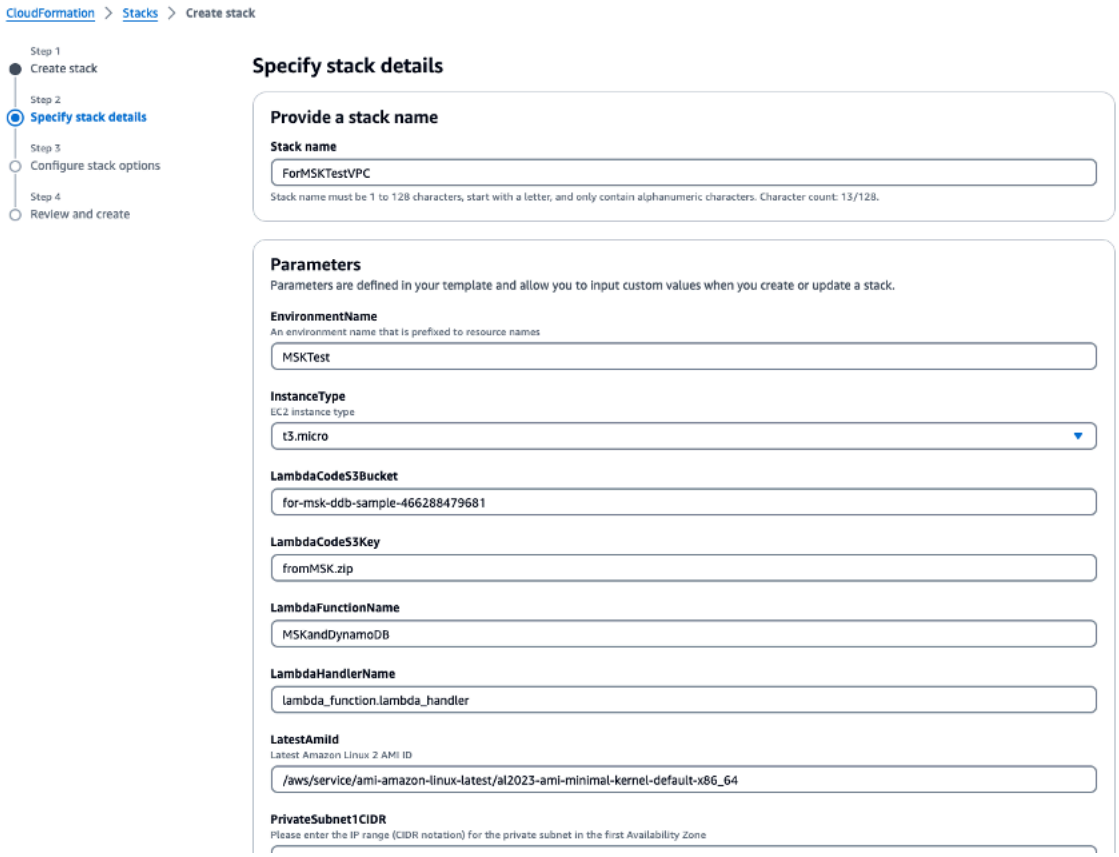
   Une fois cette opération terminée, notez le nom du compartiment S3 affiché sous Sorties. Vous aurez besoin du nom à l'étape 3.



2. Chargez le fichier ZIP `fromMSK.zip` téléchargé dans le compartiment S3 que vous venez de créer.

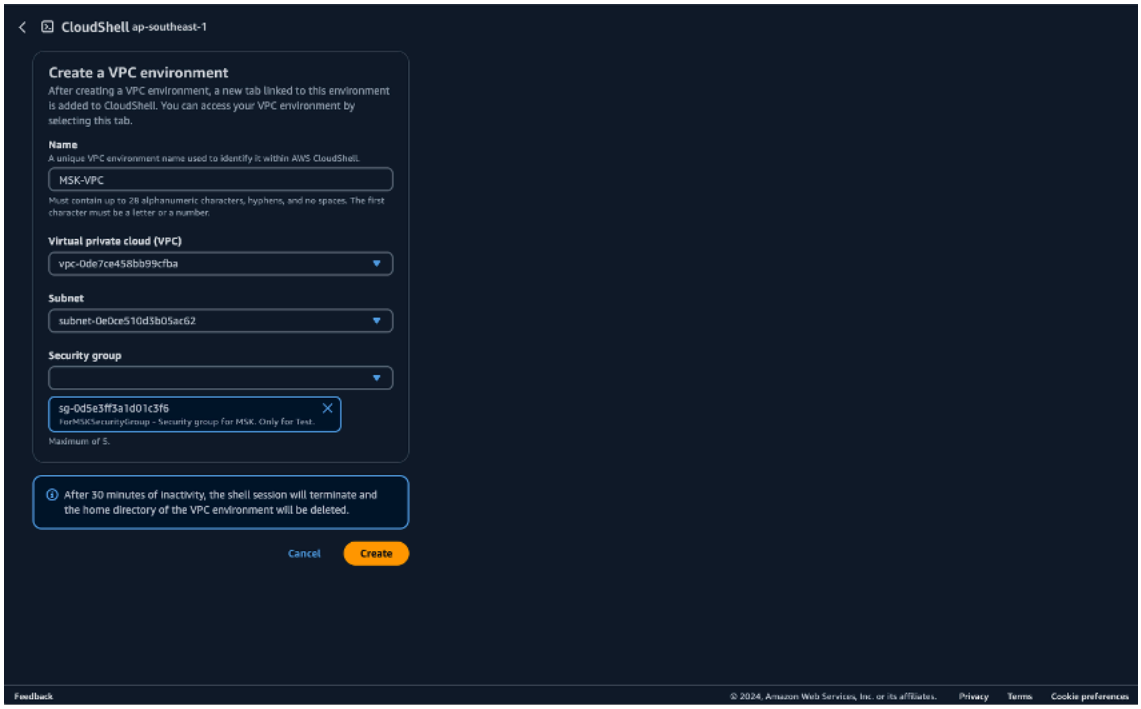


3. Exécutez le modèle CloudFormation `VPC.yaml` pour créer un VPC, un cluster Amazon MSK et une fonction Lambda. Sur l'écran de saisie des paramètres, entrez le nom du compartiment S3 que vous avez créé à l'étape 1 où il est demandé le compartiment S3. Définissez le nom de la pile CloudFormation sur `ForMSKTestVPC`.



4. Préparez l'environnement pour exécuter des scripts Python dans CloudShell. Vous pouvez utiliser CloudShell sur la AWS Management Console. Pour plus d'informations sur l'utilisation de CloudShell, consultez Getting started with AWS CloudShell. Après avoir démarré CloudShell, créez un CloudShell qui appartient au VPC que vous venez de créer afin de vous connecter au cluster Amazon MSK. Créez le CloudShell dans un sous-réseau privé. Remplissez les champs suivants :

    1. Name : peut être défini sur n'importe quel nom. MSK-VPC en est un exemple
    2. VPC : sélectionnez MSKTest
    3. Subnet : sélectionnez MSKTest Private Subnet (AZ1)
    4. SecurityGroup : sélectionnez ForMSKSecurityGroup



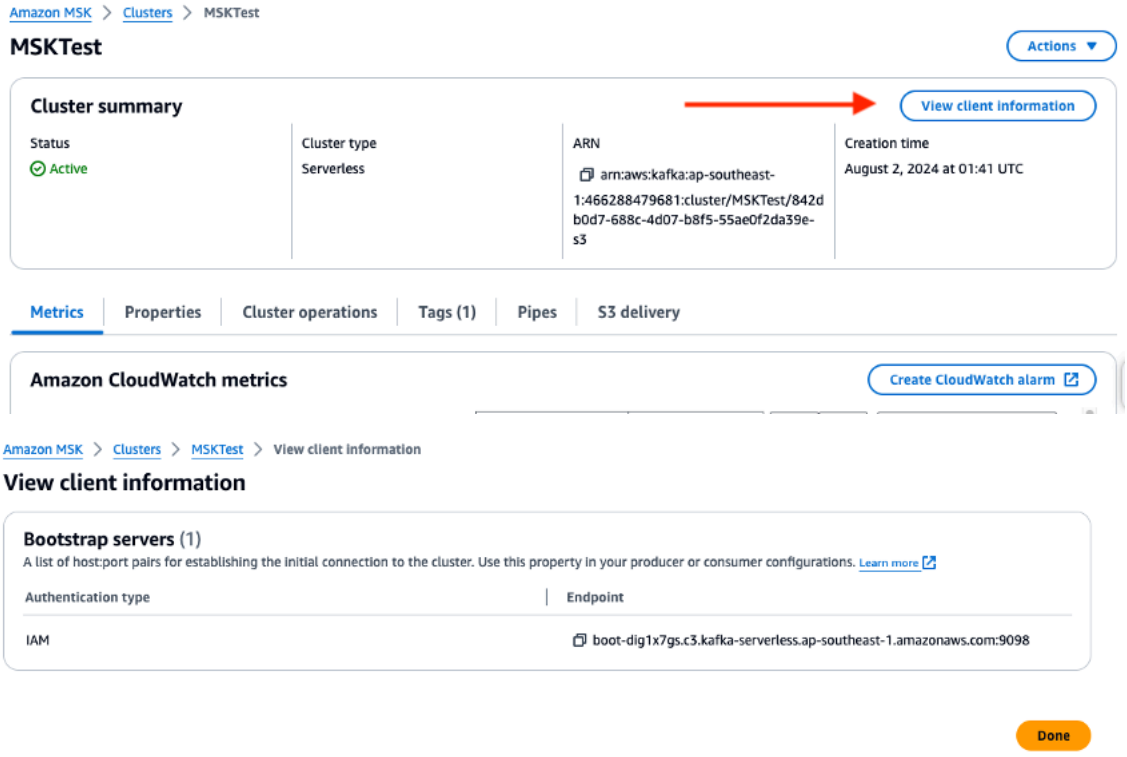
Une fois que le CloudShell appartenant au sous-réseau privé a démarré, exécutez la commande suivante :

```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Téléchargez des scripts Python à partir du compartiment S3.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Vérifiez la console de gestion et définissez les variables d'environnement pour l'URL du courtier et la valeur de région dans les scripts Python. Vérifiez le point de terminaison du courtier de clusters Amazon MSK dans la console de gestion.



7. Définissez les variables d'environnement sur le CloudShell. Si vous utilisez USA Ouest (Oregon) :

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Exécutez les script Python suivants.

   Créez une rubrique Amazon MSK.

```
python ./createTopic.py
```

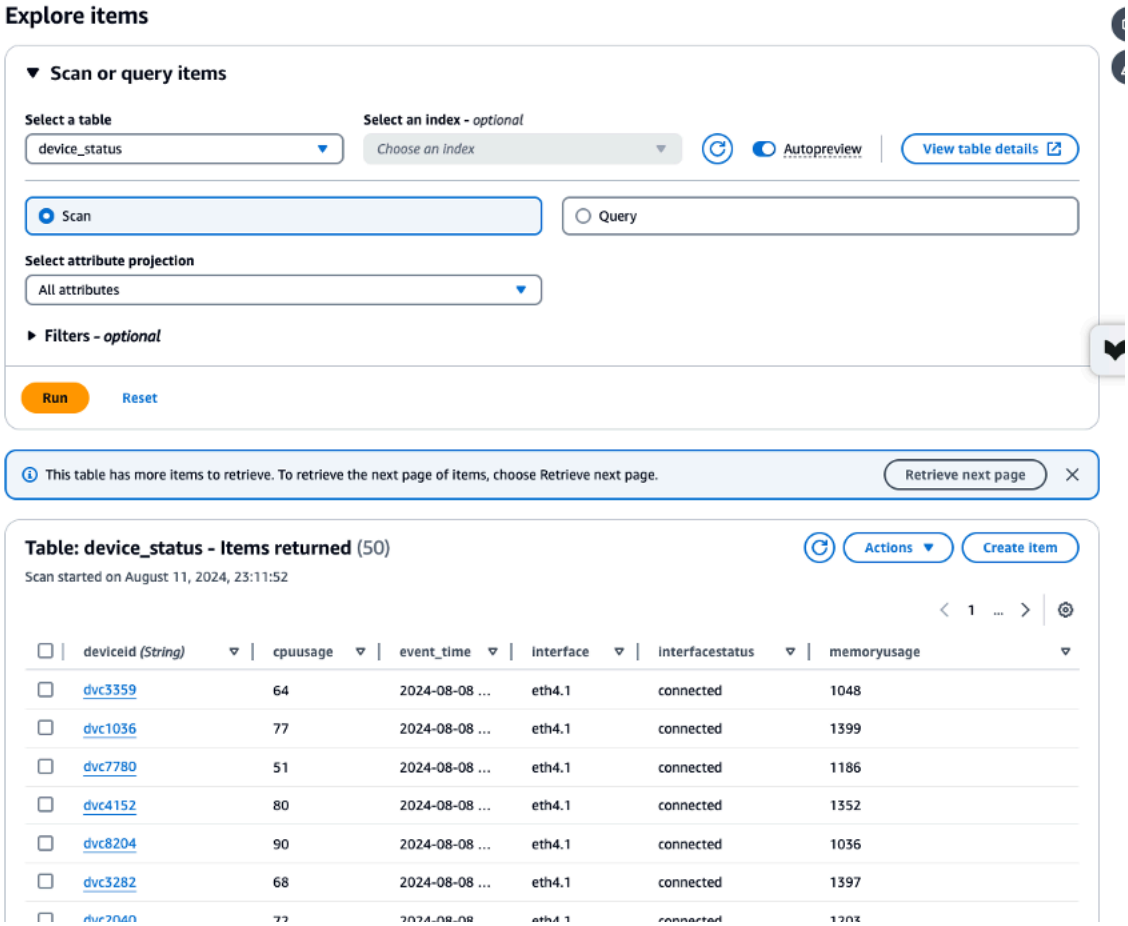
   Créez une table DynamoDB.

```
python ./createTable.py
```

Écrivez des données de test dans la rubrique Amazon MSK :

```
python ./kafkaDataGen.py
```

9. Vérifiez les métriques CloudWatch pour les ressources Amazon MSK, Lambda et DynamoDB créées, et vérifiez les données stockées dans la table `device_status` à l'aide de l'explorateur de données DynamoDB pour vous assurer que tous les processus se sont exécutés correctement. Si chaque processus est exécuté sans erreur, vous pouvez vérifier que les données de test écrites depuis CloudShell vers Amazon MSK sont également écrites sur DynamoDB.



10. Lorsque vous avez terminé avec cet exemple, supprimez les ressources créées dans ce didacticiel. Supprimez les deux piles CloudFormation : `ForMSKTestS3` et `ForMSKTestVPC`. Si la suppression de la pile aboutit, toutes les ressources seront supprimées.

Étapes suivantes

> **Note**
>
> Si vous avez créé des ressources en suivant cet exemple, pensez à les supprimer pour éviter des frais imprévus.

L'intégration a identifié une architecture qui lie Amazon MSK et DynamoDB afin de permettre aux données de flux de prendre en charge les charges de travail OLTP. À partir de là, des recherches plus complexes peuvent être effectuées en liant DynamoDB à OpenSearch Service. Envisagez d'intégrer EventBridge pour des besoins plus complexes liés à des événements, et des extensions telles que le service géré Amazon pour Apache Flink pour un débit plus élevé et des exigences de latence plus faibles.

Bonnes pratiques relatives à l'intégration à DynamoDB

Lorsque vous intégrez DynamoDB à d'autres services, vous devez toujours suivre les bonnes pratiques relatives à l'utilisation de chaque service individuel. Cependant, certaines bonnes pratiques spécifiques à l'intégration doivent être prises en compte.

Rubriques

- Création d'un instantané dans DynamoDB
- Capture des données modifiées dans DynamoDB

Création d'un instantané dans DynamoDB

- En général, nous recommandons d'utiliser l'exportation vers Amazon S3 pour créer des instantanés destinés à la réplication initiale. Cette méthode est à la fois rentable et n'entre pas en concurrence avec le trafic de votre application en matière de débit. Vous pouvez également envisager une sauvegarde et une restauration sur une nouvelle table suivies d'une opération d'analyse. Cela vous évitera de concurrencer votre application en matière de débit, mais sera généralement nettement moins rentable qu'une exportation.
- Définissez toujours un `StartTime` lorsque vous effectuez une exportation. Cela vous permet de déterminer facilement par où commencer votre capture des données modifiées (CDC).

- Lorsque vous utilisez l'exportation vers S3, définissez une action de cycle de vie sur le compartiment S3. Généralement, une action d'expiration fixée à 7 jours est sûre, mais vous devez suivre toutes les directives de votre entreprise. Même si vous supprimez explicitement vos éléments après ingestion, cette action peut permettre de détecter des problèmes, afin de réduire les coûts inutiles et d'éviter les violations des politiques.

Capture des données modifiées dans DynamoDB

- Si vous avez besoin d'une CDC en temps quasi réel, utilisez DynamoDB Streams ou Amazon Kinesis Data Streams (KDS). Lorsque vous décidez d'utiliser l'un ou l'autre de ces services, vous devez généralement prendre en compte celui qui est le plus facile à utiliser avec le service en aval. Si vous avez besoin de fournir un traitement d'événement dans l'ordre au niveau d'une clé de partition ou si vous avez un nombre d'éléments exceptionnellement important, utilisez DynamoDB Streams.
- Si vous n'avez pas besoin d'une CDC en temps quasi réel, vous pouvez utiliser l'exportation vers Amazon S3 avec des exportations incrémentielles pour exporter uniquement les modifications survenues entre deux points dans le temps.

  Si vous avez utilisé l'exportation vers S3 pour générer un instantané, cela peut vous être particulièrement utile, car vous pouvez utiliser un code similaire pour traiter les exportations incrémentielles. En général, l'exportation vers S3 est légèrement moins chère que les options de streaming précédentes, mais le coût n'est généralement pas le principal facteur déterminant pour choisir l'option à utiliser.
- Vous ne pouvez généralement avoir que deux consommateurs simultanés d'un flux DynamoDB. Tenez-en compte lors de la planification de votre stratégie d'intégration.
- N'utilisez pas d'analyses pour détecter des modifications. Cela peut fonctionner à petite échelle, mais devient peu pratique assez rapidement.

Utilisation de l’IA générative avec DynamoDB

Amazon DynamoDB est une base de données NoSQL distribuée, entièrement gérée et sans serveur offrant des performances à un chiffre en millisecondes, quelle que soit l'échelle. DynamoDB est optimisé pour les charges de travail à haut débit et vous pouvez étendre ses fonctionnalités en l’intégrant à des modèles d’IA générative. À l’aide de modèles d’IA générative, vous pouvez travailler avec les données stockées dans des tables DynamoDB en temps réel et créer des applications adaptées au contexte et hautement personnalisées. Vous pouvez également améliorer l’expérience de l’utilisateur final en exploitant pleinement les données de votre entreprise, de vos utilisateurs et de vos applications pour personnaliser vos solutions d’IA générative.

Pour plus d'informations sur l'IA de génération et les solutions proposées AWS pour créer des applications d'IA de génération, consultez Transformez votre entreprise grâce à l'IA générative.

Rubriques

- Cas d’utilisation de l’IA générative pour DynamoDB
- Blogs sur l’IA générative pour DynamoDB
- Tirer parti de l'intégration de DynamoDB Zero-ETL avec Service OpenSearch

Cas d’utilisation de l’IA générative pour DynamoDB

DynamoDB est largement utilisé dans les applications conversationnelles basées sur l’IA, telles que les chatbots et les centres d’appels basés sur un modèle de fondation (FM). Vous pouvez y accéder FMs via Amazon Bedrock, Amazon SageMaker AI ou d'autres fournisseurs de modèles. Ces applications utilisent généralement DynamoDB pour améliorer la personnalisation et l’expérience utilisateur selon trois modèles de données : les données d’application, les données métier et les données utilisateur. Voici quelques exemples de ces modèles de données :

- Stockage des données d'application, telles que l'historique des messages de chat, par le biais d'LangChainintégrations ou d'un code personnalisé. LlamaIndex Ce contexte améliore l’expérience utilisateur en permettant au modèle de converser avec l’utilisateur de façon bidirectionnelle.
- Création d’une expérience utilisateur personnalisée en exploitant les données métier, telles que l’inventaire, les prix et la documentation.
- Application des données utilisateur, telles que l’historique Web, les commandes passées et les préférences des utilisateurs, pour fournir des réponses personnalisées.

Par exemple, une compagnie d'assurance peut créer un chatbot à l'aide de DynamoDB pour permettre à son modèle d'IA basé sur la génération à enrichissement contextuel (RAG) d'accéder à des données en temps quasi réel. Des exemples de telles données sont les taux hypothécaires en temps réel, le prix des produits, la copie compliant/standard du contrat, l'historique Web des utilisateurs et les préférences des utilisateurs. La combinaison de DynamoDB et de RAG permet d'obtenir des informations détaillées et actualisées sur les produits d'assurance et les données utilisateur. Cela enrichit les invites et les réponses pour offrir aux utilisateurs finaux une expérience précise, personnalisée et en temps quasi réel.

De même, les clients du secteur des services financiers utilisent DynamoDB, les bases de connaissance Amazon Bedrock et les agents Amazon Bedrock pour créer des applications d'IA générative basées sur RAG. Ces applications peuvent utiliser des rapports de revenus open source et des transcriptions d'appels. Elles peuvent également utiliser l'historique de portefeuille et de transaction spécifique à l'utilisateur pour générer un résumé du portefeuille à la demande, y compris des perspectives pour le futur.

Blogs sur l'IA générative pour DynamoDB

Les articles suivants présentent des cas d'utilisation détaillés, les meilleures pratiques et des step-by-step guides destinés à vous aider à tirer parti des capacités de DynamoDB pour créer des applications avancées basées sur l'IA.

- Modèles de données Amazon DynamoDB pour les chatbots basés sur l'IA générative
- Créez un chatbot évolutif et sensible au contexte avec Amazon DynamoDB, Amazon Bedrock et LangChain

Tirer parti de l'intégration de DynamoDB Zero-ETL avec Service OpenSearch

Vous pouvez utiliser Amazon Bedrock avec DynamoDB pour fournir un accès sans serveur aux modèles fondamentaux (FMs), tels qu'Amazon Titan et d'autres modèles tiers. Vous pouvez tirer parti de l'intégration Zero-ETL avec Amazon OpenSearch Service pour activer les fonctionnalités de recherche vectorielle lors de la création d'applications d'IA génératives. L'atelier Generative AI with DynamoDB Zero-ETL to OpenSearch integration et Amazon Bedrock vous permet d'acquérir une expérience pratique de la configuration de l'intégration de DynamoDB Zero-ETL avec. OpenSearch Cet atelier exécute les tâches suivantes :

- Crée un pipeline à partir de votre table DynamoDB vers. OpenSearch
- Crée un connecteur Amazon Bedrock dans. OpenSearch
- Interroge Amazon Bedrock en OpenSearch tant que boutique vectorielle.
- Utilise le Claude FM dans Amazon Bedrock pour créer une réponse écrite en anglais simple expliquant les résultats de recherche renvoyés par OpenSearch.

Cet atelier vous permet d'intégrer DynamoDB pour créer des applications OpenSearch d'IA génératives. Il démontre également la capacité d'interrogation flexible entre les moteurs de base de données pour vous aider à intégrer DynamoDB OpenSearch et pour les cas d'utilisation traditionnels. Cet atelier est l'un des sept modules de la journée d'immersion Amazon DynamoDB. Vous pouvez organiser cet atelier dans n'importe quel endroit Compte AWS.

Vous pouvez également consulter le billet de blog suivant pour savoir comment configurer une intégration zéro ETL entre DynamoDB OpenSearch et Service. Ce billet de blog explique également comment configurer des connecteurs de modèle dans OpenSearch Service afin de générer automatiquement des intégrations à l'aide d'Amazon Bedrock pour les données entrantes. Recherche vectorielle pour Amazon DynamoDB sans aucun ETL pour Amazon Service. OpenSearch

Quotas et contraintes pour Amazon DynamoDB

Cette rubrique décrit les quotas actuels, anciennement appelés limites, dans Amazon DynamoDB. Cette rubrique décrit également comment effectuer les tâches de gestion des quotas, par exemple en consultant vos quotas actuels et en demandant une augmentation de quota.

Rubriques

- Exécution de tâches de gestion des quotas dans DynamoDB
- Demande d'augmentation de quota dans DynamoDB
- Quotas dans Amazon DynamoDB
- Contraintes dans Amazon DynamoDB

Exécution de tâches de gestion des quotas dans DynamoDB

Amazon DynamoDB comporte plusieurs composants de service, tels que des tables, des flux, des index, etc. Lorsque vous créez votre Compte AWS, des quotas par défaut (anciennement appelés limites) sont définis pour ces composants. Sauf indication contraire, chaque quota est spécifique à la région. Vous pouvez demander une augmentation de certains de ces quotas. Une fois qu'un quota de ressource a été atteint, les demandes supplémentaires pour créer cette ressource échouent avec une exception.

Accès aux quotas de DynamoDB

Vous pouvez utiliser DynamoDB Service Quotas de l'une des façons suivantes :

- AWS Management Console

  La console Service Quotas est une interface basée sur navigateur que vous pouvez utiliser pour gérer Service Quotas. Vous pouvez accéder à Service Quotas depuis n'importe quelle page de la AWS Management Console en les sélectionnant dans la barre de navigation supérieure ou en recherchant Service Quotas dans la AWS Management Console.

- Outils AWS Command Line Interface

  Lorsque vous utilisez des outils de l'AWS Command Line Interface, vous pouvez envoyer des commandes à la ligne de commande de votre système afin d'effectuer des tâches de Service

Quotas. Les outils de ligne de commande sont utiles si vous souhaitez créer des scripts exécutant des tâches AWS.

- Kits AWS SDK

  Vous pouvez utiliser les kits AWS SDK pour différentes langages et plateformes de programmation (par exemple, Java, Python, Ruby, .NET, iOS et Android, etc.) afin d'effectuer des tâches de Service Quotas.

Si aucun quota ajustable n'est disponible dans la console Service Quotas, utilisez la AWS Support Center Console pour créer un cas d'augmentation de quotas de service.

Affichage des quotas actuels dans la console

Pour afficher les quotas DynamoDB actuels à l'aide de la console Service Quotas

1. Ouvrez la console Service Quotas à l'adresse https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Dans la barre de navigation, en haut de l'écran, sélectionnez une région.
3. La console affiche des informations sur le Nom du quota DynamoDB, la valeur de quota appliquée au niveau du compte, la valeur de quota par défaut AWS, l'Utilisation et l'Ajustabilité du quota au niveau du compte ou au niveau des ressources.

   Si la valeur du quota ou l'utilisation appliquée n'est pas disponible, la console affiche Non disponible. Vous pouvez demander la valeur de quota que vous avez appliquée via la console du Centre de support.
4. Choisissez un Nom de quota spécifique pour afficher la page Détails, qui affiche la Description, le Code de quota, l'ARN du quota, l'Utilisation, la Valeur de quota appliquée au niveau du compte, l'Ajustabilité et la Valeur de quota par défautAWS.

   Le cas échéant, la page Détails affiche également les options de Surveillance, les Alarmes, l'historique des demandes et les balises du quota.

Affichage des quotas actuels à l'aide de l'AWS CLI

Pour afficher les valeurs par défaut des quotas DynamoDB :

- Appelez l'opération `ListDefaultServiceQuotas` avec le code de service DynamoDB (dynamodb) pour récupérer les valeurs par défaut des quotas de service Amazon DynamoDB.

```
$ aws service-quotas list-aws-default-service-quotas \
        --service-code dynamodb

 {
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        },
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-AB614373",
            "QuotaCode": "L-AB614373",
            "QuotaName": "Table-level write throughput limit",
            "Value": 40000.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }......
    ]
}
```

Pour consulter les valeurs de quota appliquées :

- Appelez l'opération `ListServiceQuotas` avec le code de service DynamoDB (dynamodb) pour récupérer toutes les valeurs de quota appliquées au niveau du compte, au niveau des

ressources ou à tous les niveaux en passant `ACCOUNT`, `RESOURCE`, ou `ALL` respectivement en tant que valeur du paramètre `QuotaAppliedAtLevel`. L'exemple d'interface de ligne de commande suivant récupère les valeurs de quota appliquées au niveau du compte.

```
$ aws service-quotas list-service-quotas \
        --service-code dynamodb \
        --quota-applied-at-level ACCOUNT


{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L-
F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L
-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false,
            "QuotaAppliedAtLevel": "ACCOUNT"
        }.....

        }
    ]
}
```

Demande d’augmentation de quota dans DynamoDB

Vous pouvez demander une augmentation de quota pour chaque région à l’aide de la console Service Quotas, de l’AWS CLI ou d’un cas de support. Si aucun quota ajustable n’est disponible dans la console Service Quotas, utilisez la AWS Support Center Console pour créer un cas d’augmentation du quota de service.

Support pourrait approuver, refuser ou approuver partiellement vos demandes d’augmentation de quota. Les augmentations ne sont pas accordées immédiatement et peuvent prendre quelques jours avant d’entrer en vigueur.

Pour demander une augmentation à l’aide de la console Service Quotas

1. Ouvrez la console Service Quotas à l’adresse https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Dans la barre de navigation, en haut de l’écran, sélectionnez une région.
3. Filtrez la liste par nom de ressource. Par exemple, saisissez À la demande pour connaître les quotas des instances à la demande.
4. Si le quota est ajustable, sélectionnez-le, puis choisissez Demander une augmentation de quota.
5. Pour Modifier la valeur du quota, saisissez la nouvelle valeur du quota.
6. Choisissez Request (Demander).
7. Pour afficher les demandes en attente ou récemment résolues dans la console, choisissez Tableau de bord dans le volet de navigation. Pour les demandes en attente, choisissez l’état de la demande pour ouvrir le reçu de la demande. L’état initial d’une demande est Pending (En attente). Une fois que le statut est passé à Quota demandé, vous verrez le numéro de dossier auprès d’Support. Choisissez le numéro de dossier pour ouvrir le billet pour votre demande.

Pour plus d’informations, notamment sur l’utilisation d’AWS CLI ou des kits SDK pour demander une augmentation de quota, consultez la section Demander une augmentation de quota (français non garanti) dans le Guide de l’utilisateur Service Quotas.

Quotas dans Amazon DynamoDB

Cette section décrit les quotas actuels, anciennement appelés limites, dans Amazon DynamoDB. Chaque quota s’applique par région, sauf indication contraire.

> **Note**
>
> Toutes les mesures de taille dans DynamoDB utilisent des unités binaires. DynamoDB indique 1 Ko = 1024 octets, 1 Mo = 1024 Ko, 1 Go = 1024 Mo, 1 To = 1024 Go.

Rubriques

- Débit de lecture/écriture
- Capacités réservées
- Tables
- Tables globales
- Index secondaires
- Attributs d’index secondaire projetés
- DynamoDB Streams
- Importer des données depuis Amazon S3
- Exportation de table vers Amazon S3.
- Sauvegarde et restauration
- Contributor Insights

Débit de lecture/écriture

Quotas de débit par défaut

AWS place des quotas par défaut sur le débit que votre compte peut fournir et consommer au sein d'une région.

Les quotas de débit de lecture au niveau du compte et de débit d’écriture au niveau du compte s’appliquent au niveau du compte. Ces quotas au niveau du compte s’appliquent à la somme de la capacité de débit provisionnée pour toutes les tables de votre compte et les index secondaires globaux dans une région donnée. Tous les débits disponibles du compte peuvent être appliqués à une même table ou à plusieurs tables. Ces quotas s’appliquent uniquement aux tables utilisant le mode de capacité provisionnée.

Les quotas de débit de lecture et d'écriture au niveau de la table s'appliquent différemment aux tables qui utilisent le mode de capacité provisionné, et aux tables qui utilisent le mode de capacité à la demande.

Pour les tables en mode capacité provisionnée GSIs, le quota est le nombre maximum d'unités de capacité de lecture et d'écriture pouvant être provisionnées pour n'importe quelle table ou n'importe laquelle de ses tables GSIs dans la région. Le total de chaque table individuelle et de toutes ses composantes GSIs doit également rester inférieur au quota de débit de lecture et d'écriture au niveau du compte. Cela s'ajoute à l'exigence selon laquelle le total de toutes les tables provisionnées et leur contenu GSIs doivent rester inférieurs au quota de débit de lecture et d'écriture au niveau du compte.

Pour les tables en mode capacité à la demande et GSIs, le quota au niveau de la table correspond aux unités de capacité de lecture et d'écriture maximales disponibles pour une table, ou pour tout GSI individuel au sein de cette table. Aucun quota de débit de lecture et d'écriture au niveau du compte n'est appliqué aux tables en mode à la demande.

Vous trouverez ci-dessous les quotas de débit qui s'appliquent à votre compte, par défaut.

> **Note**
>
> Vous pouvez demander un nombre illimité d'unités de capacité de lecture (RCU) ou d'unités de capacité d'écriture (WCU) pour vos tables DynamoDB par le biais d'une augmentation du quota de service. Les valeurs répertoriées dans le tableau suivant représentent les quotas initiaux par défaut. Il ne s'agit pas de limites maximales pour vos tables.

Nom du quota de débit	À la demande	Alloué	Ajustable
Par table	40 000 unités de demande de lecture et 40 000 unités de demande d'écriture	40 000 unités de capacité en lecture et 40 000 unités de capacité en écriture	Oui
Par compte	Non applicable	80 000 unités de capacité en lecture et 80 000 unités de capacité en écriture	Oui

Nom du quota de débit	À la demande	Alloué	Ajustable
Débit minimal pour n'importe quelle table ou GSI	Non applicable	1 unité de capacité de lecture et 1 unité de capacité d'écriture	Oui

Augmentation ou diminution du débit (pour les tables allouées)

Augmentation du débit alloué

Vous pouvez accroître la `ReadCapacityUnits` ou `WriteCapacityUnits` aussi souvent que nécessaire, à l'aide de la AWS Management Console ou de l'opération `UpdateTable`. En un seul appel, vous pouvez augmenter le débit alloué pour une table, pour un index secondaire global de cette table ou pour toute combinaison de ceux-ci. Les nouveaux paramètres ne prennent pas effet jusqu'à ce que l'opération `UpdateTable` soit achevée.

Vous ne pouvez pas dépasser vos quotas par compte lorsque vous ajoutez une capacité approvisionnée, et DynamoDB ne vous permet pas d'augmenter la capacité approvisionnée très rapidement. En dehors de ces restrictions, vous pouvez augmenter la capacité allouée de vos tables autant que nécessaire. Pour de plus amples informations sur les quotas par compte, veuillez consulter la section précédente, Quotas de débit par défaut.

Diminution du débit alloué

Pour chaque table et index secondaire global dans une opération `UpdateTable`, vous pouvez diminuer `ReadCapacityUnits` ou `WriteCapacityUnits` (ou les deux). Les nouveaux paramètres ne prennent effet qu'après que l'opération `UpdateTable` est achevée.

Il existe un quota par défaut concernant le nombre de diminutions de capacité provisionnées que vous pouvez effectuer chaque jour sur votre table DynamoDB. Une journée est définie conformément à l'heure UTC (Universal Time Coordinated). Un jour donné, vous pouvez commencer par effectuer jusqu'à quatre diminutions en une heure tant que vous n'avez pas encore effectué d'autres diminutions durant cette journée. Par la suite, vous pouvez effectuer une diminution supplémentaire par heure (une fois toutes les 60 minutes). Cela porte effectivement le nombre maximum de diminutions par jour à 27 fois.

> **Important**
>
> Les limites de réduction de la table et de l'index secondaire global sont découplées. Ainsi, les index globaux secondaires d'une table particulière ont leurs propres limites de réduction. Toutefois, si une seule et même demande réduit le débit pour une table et un index secondaire global, elle est rejetée si l'une ou l'autre dépassent les limites actuelles. Les demandes ne sont pas partiellement traitées.

Example

Au cours des quatre premières heures d'une journée, une table avec un index secondaire global peut être modifiée comme suit :

- Diminuer la `WriteCapacityUnits` ou la `ReadCapacityUnits` de la table (ou les deux) quatre fois.
- Diminuer la `WriteCapacityUnits` ou la `ReadCapacityUnits` de l'index secondaire global (ou les deux) quatre fois.

À la fin de cette même journée, le débit de la table et de l'index secondaire global peut être réduit au total 27 fois pour chacun.

Capacités réservées

AWS place un quota par défaut sur la quantité de capacité réservée active que votre compte peut acheter. La limite de quota est une combinaison de capacité réservée pour les unités de capacité d'écriture (WCUs) et de capacité de lecture (RCUs).

Quota de capacité réservée	Capacité réservée active	Ajustable
Par compte	1 000 000 unités de capacité provisionnées (_) WCUs RCUs	Oui

Si vous tentez d'acheter plus de 1 000 000 d'unités de capacité provisionnée en une fois, vous recevrez un message d'erreur concernant la limite de quota de service. Si vous disposez d'une

capacité réservée active et que vous tentez d'acheter une capacité réservée supplémentaire qui se traduirait par plus de 1 000 000 d'unités de capacité provisionnée actives, vous recevrez un message d'erreur concernant la limite de quota de service.

Tables

Taille des tables

Il n'y a pas de limite pratique sur la taille d'une table. Les tables sont sans contraintes en ce qui concerne le nombre d'éléments ou le nombre d'octets.

Nombre maximal de tables par région et par compte

Quel que soit le AWS compte, il existe un quota initial de 2 500 tables par AWS région.

Si vous avez besoin de plus de 2 500 tables pour un seul compte, contactez l'équipe chargée de votre compte AWS pour explorer la possibilité d'augmenter le nombre de tables jusqu'à 10 000. Pour plus de 10 000 tables, la bonne pratique recommandée consiste à configurer plusieurs comptes, chacun pouvant servir jusqu'à 10 000 tables.

Tables globales

Les quotas par défaut suivants s'appliquent lors de l'utilisation de tables globales.

Quota de tables globales par défaut	À la demande	Alloué
Nombre de tables globales du MRSC (consultez the section called "Modes de cohérence")	400 tables globales MRSC au total, quel que soit le mode de capacité	400 tables globales MRSC au total, quel que soit le mode de capacité
Débit par table configuré pour une cohérence à terme entre plusieurs régions (MREC)	40 000 unités de demande de lecture et 40 000 unités de demande d'écriture	40 000 unités de capacité en lecture et 40 000 unités de capacité en écriture
Débit par table configuré pour une forte cohérence entre plusieurs régions (MRSC)	40 000 unités de demande de lecture et 40 000 unités de demande d'écriture	40 000 unités de capacité en lecture et 40 000 unités de capacité en écriture

Quota de tables globales par défaut	À la demande	Alloué
Données de remplacement pour les nouveaux réplica par compte, par région et par jour	10 To	10 To

> **Note**
>
> Dans certains cas, vous devrez peut-être demander une augmentation de la limite de quota AWS Support. Si vous êtes dans l'une des situations suivantes, consultez https://aws.amazon.com/support :
>
> - Les quotas de débit de tables globales doivent être égaux ou supérieurs aux quotas de débit par table pour que la création de réplica réussisse. Il existe des quotas de débit de tables globales distincts pour les tables globales MREC et MRSC.
> - Si vous ajoutez un ou plusieurs réplicas dans une Région de destination pendant une période de 24 heures avec un total combiné supérieur à 10 To, vous devez demander une augmentation du quota de service pour votre quota de remplissage de données de réplica supplémentaire.
> - Si vous obtenez une erreur similaire à ce qui suit :
>   - Impossible de créer un réplica de la table « example_table » dans la région « example_region_A » car elle dépasse la limite de votre compte actuel dans la région « example_region_B ».

Index secondaires

Vous pouvez définir jusqu'à 5 index secondaires locaux par table.

Il existe un quota par défaut de 20 index secondaires globaux par table.

Attributs d'index secondaire projetés

Vous pouvez projeter jusqu'à 100 attributs combinés pour l'ensemble des index secondaires locaux et GSI d'une table. Ce quota ne s'applique qu'aux attributs projetés spécifiés par l'utilisateur.

Pour l'opération `CreateTable`, si vous spécifiez un `ProjectionType` d'`INCLUDE`, le nombre total d'attributs spécifié dans `NonKeyAttributes`, cumulé sur l'ensemble des index secondaires, ne peut pas dépasser 100. La projection du même nom d'attribut dans deux index différents équivaut à deux attributs distincts par rapport au quota.

Ce quota ne s'applique pas aux index secondaires dont `ProjectionType` a la valeur `KEYS_ONLY` ou `ALL`.

DynamoDB Streams

Lecteurs simultanés d'une partition dans DynamoDB Streams

Pour les tables à région unique qui ne sont pas des tables globales, vous pouvez concevoir jusqu'à deux processus pour lire la même partition DynamoDB Streams simultanément. Le dépassement de cette limite peut se traduire par une limitation de la demande. Pour les tables globales, nous vous recommandons de limiter le nombre de lecteurs simultanés à un seul pour éviter la limitation des demandes.

Capacité d'écriture maximum pour une table avec DynamoDB Streams activé

AWS place des quotas par défaut sur la capacité d'écriture des tables DynamoDB lorsque DynamoDB Streams est activé. Ces quotas par défaut ne s'appliquent qu'aux tables en mode read/write capacité allouée.

- Par table – 40 000 unités de capacité d'écriture

Importer des données depuis Amazon S3

L'importation vers DynamoDB depuis Amazon S3 peut prendre en charge jusqu'à 50 tâches d'importation simultanées avec une taille d'objet d'importation totale de 15 To à la fois dans les régions us-east-1, us-west-2, us-west-1. Dans toutes les autres régions, jusqu'à 50 tâches d'importation simultanées d'une taille totale de 1 To sont prises en charge. Chaque tâche d'importation peut prendre jusqu'à 50 000 objets Amazon S3 dans toutes les régions. Pour plus d'informations sur l'importation et la validation, consultez la section Quotas de format d'importation et validation.

Exportation de table vers Amazon S3.

Exportation complète : permet d'exporter jusqu'à 300 tâches d'exportation simultanées ou un volume total de 100 To pour l'ensemble des exportations de tables en cours. Ces deux limites sont vérifiées avant qu'une exportation ne soit mise en file d'attente.

Exportation incrémentielle : l'exportation incrémentielle de DynamoDB vers Amazon S3 peut prendre en charge jusqu'à 300 tâches d'exportation simultanées ou un volume total de 100 To pour l'ensemble des exportations de tables en vol. Les limites de la fenêtre de période d'exportation sont de 15 minutes minimum et 24 heures maximum.

Sauvegarde et restauration

DynamoDB prend en charge jusqu'à 50 restaurations simultanées pour un total de 50 To via des sauvegardes DynamoDB à la demande ou en continu.AWS Backup prend en charge jusqu'à 50 restaurations simultanées pour un total de 25 To.

Contributor Insights

Lorsque vous activez Customer Insights sur votre table DynamoDB, vous êtes toujours soumis aux limites des règles de Contributor Insights. Pour plus d'informations, consultez Quotas de service CloudWatch .

Contraintes dans Amazon DynamoDB

Cette section décrit les contraintes actuelles, anciennement appelées limites, dans Amazon DynamoDB.

> **Note**
>
> Toutes les mesures de taille dans DynamoDB utilisent des unités binaires. DynamoDB indique 1 Ko = 1024 octets, 1 Mo = 1024 Ko, 1 Go = 1024 Mo, 1 To = 1024 Go.

Rubriques

- Mode de capacité en lecture/écriture
- Index secondaires
- Clés de partition et clés de tri

- Règles de dénomination
- Types de données
- Éléments
- Attributes
- Paramètre d'expression
- Transactions DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- Contraintes propres à l'API
- Chiffrement de DynamoDB au repos

Mode de capacité en lecture/écriture

Vous pouvez faire passer les tables du mode de capacité provisionnée au mode à la demande jusqu'à quatre fois par période de 24 heures. Vous pouvez à tout moment faire passer des tables du mode à la demande au mode de capacité provisionnée.

Pour plus d'informations sur le basculement entre les modes de capacité de lecture et d'écriture, consultez Considérations relatives au changement de mode de capacité dans DynamoDB.

Tailles des unités de capacité (pour les tables allouées)

Une unité de capacité de lecture équivaut à une lecture cohérente forte par seconde, ou deux lectures éventuellement cohérentes par seconde d'éléments dont la taille peut atteindre 4 Ko.

Une unité de capacité d'écriture équivaut à une écriture par seconde d'éléments dont la taille peut atteindre 1 Ko.

Les demandes de lecture transactionnelles nécessitent deux unités de capacité de lecture pour effectuer une lecture par seconde d'éléments dont la taille peut atteindre 4 Ko.

Les demandes d'écriture transactionnelles nécessitent deux unités de capacité de lecture pour effectuer une lecture par seconde d'éléments dont la taille peut atteindre 1 Ko.

Tailles des unités de demande (pour les tables à la demande)

Une unité de demande de lecture équivaut à une lecture fortement cohérente par seconde, ou deux lectures éventuellement cohérentes par seconde d'éléments dont la taille peut atteindre 4 Ko.

Une unité de demande d'écriture équivaut à une écriture par seconde, pour les éléments dont la taille peut atteindre 1 Ko.

Les demandes de lecture transactionnelles nécessitent deux unités de demande de lecture pour effectuer une lecture par seconde d'éléments dont la taille peut atteindre 4 Ko.

Les demandes d'écriture transactionnelles nécessitent deux unités de demande d'écriture pour effectuer une écriture par seconde d'éléments dont la taille peut atteindre 1 Ko.

Index secondaires

Attributs d'index secondaire projeté par table

Vous pouvez projeter jusqu'à un total de 100 attributs dans l'ensemble des index secondaires locaux et globaux d'une table. Cette possibilité ne s'applique qu'aux attributs projetés spécifiés par l'utilisateur.

Dans une opération `CreateTable`, si vous spécifiez un `ProjectionType` de `INCLUDE`, le nombre total d'attributs spécifié dans `NonKeyAttributes`, cumulé sur l'ensemble des index secondaires, ne peut pas dépasser 100. Si vous projetez le même nom d'attribut dans deux index différents, cette projection est considérée comme deux attributs distincts lors de la détermination du total.

Cette limite ne s'applique pas aux index secondaires dont `ProjectionType` a la valeur `KEYS_ONLY` ou `ALL`.

Clés de partition et clés de tri

Longueur de clé de partition

La longueur minimale d'une valeur de clé de partition est 1 octet. La longueur maximale est de 2 048 octets.

Valeurs de clé de partition

Il n'existe pas de limite pratique quant au nombre de valeurs de clé de partition distinctes, pour les tables ou pour les index secondaires.

Longueur de la clé de tri

La longueur minimale d'une valeur de clé de tri est 1 octet. La longueur maximale est de 1 024 octets.

Valeurs de clé de tri

En général, il n'existe pas de limite pratique sur le nombre de valeurs de clé de tri distinctes par valeur de clé de partition.

L'exception concerne les tables avec index secondaires. Une collection d'éléments est l'ensemble des éléments qui ont la même valeur d'attribut de clé de partition. Dans un index secondaire global, la collection d'éléments est indépendante de la table de base (et peut avoir un attribut de clé de partition différent), mais dans un index secondaire local, la vue indexée est colocalisée dans la même partition que l'élément de la table et partage le même attribut de clé de partition. En raison de cette localité, lorsqu'une table en possède une ou plusieurs LSIs, la collection d'éléments ne peut pas être distribuée sur plusieurs partitions.

Pour une table comportant un ou plusieurs articles LSIs, la taille des collections d'articles ne peut pas dépasser 10 Go. Cela inclut tous les éléments de la table de base et toutes les vues d'index secondaires locaux (LSI) projetées ayant la même valeur d'attribut de clé de partition. 10 Go est la taille maximale d'une partition. Pour en savoir plus, consultez Taille limite de collection d'éléments.

Règles de dénomination

Noms de tables et d'index secondaires

Les noms de table et les noms d'index secondaires doivent comporter au moins 3 caractères, mais ne pas dépasser 255 caractères. Voici les caractères acceptés :

- `A-Z`
- `a-z`
- `0-9`
- `_` (soulignement)
- `-` (trait d'union)
- `.` (point)

Noms d'attribut

En général, un nom d'attribut doit comporter au moins un caractère, mais ne pas dépasser 64 Ko.

Les éléments suivants sont les exceptions. Ces noms d'attribut ne doivent pas dépasser 255 caractères :

- Noms de clés de partition d'index secondaire.
- Noms de clés de tri d'index secondaire.
- Noms d'attributs projetés spécifiés par l'utilisateur (applicables uniquement aux index secondaires locaux). Dans une opération `CreateTable`, si vous spécifiez un `ProjectionType` de `INCLUDE`, les noms des attributs du paramètre `NonKeyAttributes` sont restreints en longueur. Les types de projection `KEYS_ONLY` et `ALL` ne sont pas concernés.

Ces noms d'attribut doivent être codés en UTF-8 et la taille totale de chaque nom (après encodage) ne peut pas dépasser 255 octets.

Types de données

String

La longueur d'une chaîne est limitée par la taille maximum d'élément de 400 Ko.

Les chaînes sont Unicode avec codage binaire UTF-8. L'encodage UTF-8 étant de largeur variable, DynamoDB détermine la longueur d'une chaîne à l'aide de ses octets UTF-8.

Number

Un nombre peut avoir jusqu'à 38 chiffres de précision et peut être positif, négatif ou nul.

- Plage positive : `1E-130` à `9.99999999999999999999999999999999999999E+125`
- Plage négative : `-9.99999999999999999999999999999999999999E+125` à `-1E-130`

DynamoDB utilise des chaînes JSON pour représenter les données de type Number (nombre) dans les demandes et les réponses. Pour de plus amples informations, veuillez consulter API de bas niveau de DynamoDB.

Si la précision numérique est importante, passez les nombres à DynamoDB à l'aide de chaînes que vous convertissez à partir de données de type Number.

Binaire

La longueur d'un élément de type Binary (binaire) est limitée par la taille maximum d'élément de 400 Ko.

Les applications qui fonctionnent avec des attributs de type Binary doivent encoder les données au format base 64 avant de les envoyer à DynamoDB. À la réception des données, DynamoDB les décode dans un tableau d'octets non signés, et utilise ces informations comme longueur de l'attribut.

Éléments

Taille de l'élément

La taille maximum d'un élément dans DynamoDB est de 400 Ko, incluant la longueur binaire du nom d'attribut (longueur UTF-8) et les longueurs de valeur d'attribut (également binaires). Le nom d'attribut est comptabilisé parmi la limite de taille.

Par exemple, imaginons un élément avec deux attributs : un attribut nommé « shirt-color » avec la valeur « R » et un autre attribut nommé « shirt-size » avec la valeur « M ». La taille totale de cet élément est de 23 octets.

Taille d'élément pour les tables avec des index secondaires locaux

Pour chaque index secondaire local sur une table, il existe une limite de 400 Ko au total des éléments suivants :

- La taille des données d'un élément de la table.
- La taille des entrées correspondantes (y compris les valeurs de clé et les attributs projetés) dans tous les index secondaires locaux.

Attributes

Paires nom-valeur d'attribut par élément

La taille cumulée des attributs par élément ne peut pas dépasser la taille maximum d'élément DynamoDB (400 Ko).

Nombre de valeurs dans un type list, map ou set

Il n'existe aucune limite au nombre de valeurs dans un élément de type List (liste), Map (mappage) ou Set (ensemble), pour autant que la taille de l'élément ne dépasse pas la limite de taille d'élément de 400 Ko.

Valeurs d'attribut

Les valeurs vides de String et de Binary attribute sont autorisées si l'attribut n'est pas utilisé comme attribut de clé pour une table ou un index. Les valeurs de chaîne (String) et binaires (Binary) vides sont autorisées dans les types Set (ensemble), List (liste) et Map (carte). Une valeur d'attribut ne peut pas être un ensemble vide (ensemble de chaînes, de chiffres ou binaire). Cependant, les éléments de type liste et carte vides sont autorisés.

Profondeur des attributs imbriqués

DynamoDB prend en charge les attributs imbriqués jusqu'à 32 niveaux de profondeur.

Paramètre d'expression

Les paramètres d'expression incluent `ProjectionExpression`, `ConditionExpression`, `UpdateExpression` et `FilterExpression`.

Longueurs

La longueur maximale d'une chaîne d'expression est 4 Ko. Par exemple, la taille de `ConditionExpression a=b` est de trois octets.

La longueur maximale de n'importe quel nom d'attribut d'expression ou de valeur d'attribut expression est 255 octets. Par exemple, `#name` fait 5 octets et `:val` fait 4 octets.

La longueur maximale de toutes les variables de substitution d'une expression est de 2 Mo. Il s'agit de la somme des longueurs de tous les `ExpressionAttributeNames` et `ExpressionAttributeValues`.

Opérateurs et opérandes

Le nombre maximal d'opérateurs ou de fonctions autorisés dans une `UpdateExpression` est 300. Par exemple, UpdateExpression`SET a = :val1 + :val2 + :val3`contient deux opérateurs `+` « ».

Le nombre maximal d'opérandes pour le comparateur `IN` est 100.

Mots réservés

DynamoDB ne vous empêche pas d'utiliser des noms en conflit avec des mots réservés. (Pour obtenir la liste complète, consultez Mots réservés dans DynamoDB.)

Cependant, si vous utilisez un mot réservé dans un paramètre d'expression, vous devez également spécifier `ExpressionAttributeNames`. Pour de plus amples informations, veuillez consulter Noms d'attributs d'expression (alias) dans DynamoDB.

Transactions DynamoDB

Les opérations d'API transactionnelles de DynamoDB sont sujettes aux contraintes suivantes :

- Une transaction ne peut pas contenir plus de 100 éléments uniques.
- Une transaction ne peut pas contenir plus de 4 Mo de données.
- Deux actions d'une transaction ne peuvent pas porter sur le même élément de la même table. Par exemple, vous ne pouvez pas effectuer un `ConditionCheck` et un `Update` sur le même élément dans une même transaction.
- Une transaction ne peut pas être effectuée sur les tables de plusieurs AWS comptes ou régions.
- Les opérations transactionnelles fournissent des garanties d'atomicité, de cohérence, d'isolation et de durabilité (ACID) uniquement dans la AWS région où l'écriture a été effectuée à l'origine. Les transactions ne sont pas prises en charge entre les régions dans les tables globales. Par exemple, supposons que vous disposiez d'une table globale avec des réplicas dans les régions USA Est (Ohio) et USA Ouest (Oregon) et que vous effectuiez une opération `TransactWriteItems` dans la région USA Est (Virginie du Nord). Dans ce cas, vous pouvez observer des transactions partiellement terminées dans la région USA Ouest (Oregon) lorsque les modifications sont répliquées. Les modifications seront uniquement répliquées sur les autres régions une fois validées dans la région source.

DynamoDB Streams

Lecteurs simultanés d'une partition dans DynamoDB Streams

Pour les tables à région unique qui ne sont pas des tables globales, vous pouvez concevoir jusqu'à deux processus pour lire la même partition DynamoDB Streams simultanément. Le dépassement de cette limite peut se traduire par une limitation de la demande. Pour les tables globales, nous vous recommandons de limiter le nombre de lecteurs simultanés à un seul pour éviter la limitation des demandes.

DynamoDB Accelerator (DAX)

AWS Disponibilité de la région

Pour obtenir la liste des AWS régions dans lesquelles le DAX est disponible, voir DynamoDB Accelerator (DAX) dans le. Références générales AWS

Nœuds

Un cluster DAX comporte exactement un nœud primaire et de 0 à 9 nœuds de réplica en lecture.

Le nombre total de nœuds (par AWS compte) ne peut pas dépasser 50 dans une seule AWS région.

Groupes de paramètres

Vous pouvez créer jusqu'à 20 groupes de paramètres DAX par région.

Groupes de sous-réseaux

Vous pouvez créer jusqu'à 50 groupes de sous-réseaux DAX par région.

Dans un même groupe de sous-réseaux, vous pouvez définir jusqu'à 20 sous-réseaux.

> **Important**
>
> Un cluster DAX prend en charge un maximum de 500 tables DynamoDB. Une fois que vous dépassez les 500 tables DynamoDB, la disponibilité et les performances de votre cluster peuvent se dégrader.

Contraintes propres à l'API

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

En général, vous pouvez avoir jusqu'à 500 CreateTable,, UpdateTableDeleteTablePutResourcePolicy, et DeleteResourcePolicydemandes exécutées simultanément dans n'importe quelle combinaison. En d'autres termes, le nombre total de tables ayant l'état `CREATING`, `UPDATING` ou `DELETING` ne peut pas dépasser 500.

Vous pouvez soumettre jusqu'à 2 500 demandes par seconde de demandes d'API mutables (`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy` et

DeleteResourcePolicy) du plan de contrôle, quelle que soit leur combinaison sur un groupe de tables. Cependant, les demandes PutResourcePolicy et DeleteResourcePolicy ont des limites individuelles inférieures. Pour plus d'informations, consultez les détails des quotas suivants pour PutResourcePolicy et DeleteResourcePolicy.

Le demandes CreateTable et PutResourcePolicy qui incluent une politique basée sur les ressources seront considérées comme deux demandes supplémentaires pour chaque Ko de la politique. Par exemple, une demande CreateTable ou PutResourcePolicy avec une politique de 5 Ko comptera pour 11 demandes. 1 pour la demande CreateTable et 10 pour la politique basée sur les ressources (2 x 5 Ko). De même, une politique de 20 Ko comptera pour 41 demandes. 1 pour la demande CreateTable et 40 pour la politique basée sur les ressources (2 x 20 Ko).

PutResourcePolicy

Vous pouvez envoyer jusqu'à 25 demandes d'API PutResourcePolicy par seconde sur un groupe de tables. Après une demande réussie pour une table individuelle, aucune nouvelle demande PutResourcePolicy n'est prise en charge pendant les 15 secondes suivantes.

La taille maximale prise en charge pour un document de politique basé sur les ressources est de 20 Ko. DynamoDB compte les espaces blancs lors du calcul de la taille d'une politique au regard de cette limite.

DeleteResourcePolicy

Vous pouvez envoyer jusqu'à 50 demandes d'API DeleteResourcePolicy par seconde sur un groupe de tables. Après une demande PutResourcePolicy réussie pour une table individuelle, aucune nouvelle demande DeleteResourcePolicy n'est prise en charge pendant les 15 secondes suivantes.

BatchGetItem

Une seule opération BatchGetItem peut extraire au maximum 100 éléments. La taille totale de tous les éléments extraits ne peut pas dépasser 16 Mo.

BatchWriteItem

Une seule opération BatchWriteItem peut contenir jusqu'à 25 demandes PutItem ou DeleteItem. La taille totale de tous les éléments écrits ne peut pas dépasser 16 Mo.

DescribeStream

Vous pouvez appeler `DescribeStream` au maximum 10 fois par seconde.

DescribeTableReplicaAutoScaling

La méthode `DescribeTableReplicaAutoScaling` ne prend en charge que 10 demandes par seconde.

DescribeLimits

`DescribeLimits` ne doit pas être appelé régulièrement. Vous pouvez vous attendre à des erreurs de limitation si vous l'appelez plusieurs fois par minute.

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

`DescribeContributorInsights`, `ListContributorInsights` et `UpdateContributorInsights` ne doivent être appelés que périodiquement. DynamoDB prend en charge jusqu'à cinq requêtes par seconde pour chacune de ces requêtes. APIs

DescribeTable/ListTables/GetResourcePolicy

Vous pouvez soumettre jusqu'à 2 500 demandes par seconde d'une combinaison de demandes d'API en lecture seule (`DescribeTable`, `ListTables` et `GetResourcePolicy`) du plan de contrôle. L'API `GetResourcePolicy` a une limite individuelle inférieure de 100 demandes par seconde.

DescribeTimeToLive

L'opération `DescribeTimeToLive` est limitée à 10 unités de demande de lecture par seconde. Si vous dépassez cette limite, DynamoDB renvoie une erreur `ThrottlingException`.

Query

L'ensemble de résultats d'une opération `Query` est limité à 1 Mo par appel. Vous pouvez utiliser `LastEvaluatedKey` à partir de la réponse de la requête pour récupérer plus de résultats.

`Scan`

L'ensemble de résultats d'une opération `Scan` est limité à 1 Mo par appel. Vous pouvez utiliser `LastEvaluatedKey` à partir de la réponse de l'opération Scan pour récupérer plus de résultats.

`UpdateKinesisStreamingDestination`

Lorsque vous effectuez des opérations `UpdateKinesisStreamingDestination`, vous pouvez définir `ApproximateCreationDateTimePrecision` sur une nouvelle valeur au maximum 3 fois par période de 24 heures.

`UpdateTableReplicaAutoScaling`

La méthode `UpdateTableReplicaAutoScaling` ne prend en charge que dix demandes par seconde.

`UpdateTableTimeToLive`

La méthode `UpdateTableTimeToLive` prend en charge une seule demande d'activation ou de désactivation `Time to Live (TTL)` par table spécifiée et par heure. Le traitement complet de la modification peut prendre jusqu'à une heure. Tout `UpdateTimeToLive` appel supplémentaire pour la même table pendant cette durée d'une heure entraîne un ValidationException.

Chiffrement de DynamoDB au repos

Vous pouvez basculer entre une Clé détenue par AWSClé gérée par AWS, une clé et une clé gérée par le client jusqu'à quatre fois, à tout moment par fenêtre de 24 heures, par table, à partir du moment où la table a été créée. Et si aucune modification n'est intervenue au cours des six dernières heures, une modification supplémentaire est autorisée. Cela porte ainsi le nombre maximum de modifications par jour à huit (quatre modifications au cours des six premières heures, puis une modification pour chacune des trois fenêtres de six heures suivantes).

Vous pouvez changer de clé de chiffrement pour utiliser une Clé détenue par AWS aussi souvent que nécessaire, même si le quota ci-dessus est dépassé.

Voici les quotas, à moins que vous ne demandiez un seuil supérieur. Pour demander une augmentation du quota de service, consultez https://aws.amazon.com/support.

Référence de l’API DynamoDB

La référence de l’API Amazon DynamoDB contient une liste complète des opérations prises en charge par :

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Résolution des problèmes liés à Amazon DynamoDB

Les rubriques suivantes fournissent des conseils de résolution des erreurs et des problèmes que vous pouvez rencontrer en utilisant Amazon DynamoDB. Si vous rencontrez un problème qui n'est pas répertorié ici, vous pouvez utiliser le bouton de commentaire sur cette page pour le signaler.

Pour plus de conseils de dépannage et de réponses aux questions courantes de support, visitez le Centre de connaissances AWS.

Rubriques

- Résolution des problèmes liés aux erreurs internes du serveur dans Amazon DynamoDB
- Résolution des problèmes de latence dans Amazon DynamoDB
- Résolution des problèmes de limitation dans Amazon DynamoDB

Résolution des problèmes liés aux erreurs internes du serveur dans Amazon DynamoDB

Dans DynamoDB, les erreurs internes du serveur (erreurs 500) indiquent que le service n'est pas en mesure de traiter la demande. Ces erreurs peuvent survenir pour diverses raisons, telles que des problèmes passagers de réseau dans la flotte, des problèmes d'infrastructure, des problèmes liés aux nœuds de stockage, etc.

Il est possible que vous rencontriez des erreurs internes du serveur pendant le cycle de vie de la table DynamoDB. Cela est attendu en raison de la nature distribuée du service et ne devrait généralement pas être une source de préoccupation. DynamoDB répare et résout automatiquement tout problème passager lié au service en temps réel, sans aucune intervention de votre part. Toutefois, si vous observez un nombre constamment élevé d'erreurs internes du serveur lors des demandes adressées à la table (comme le montre la métrique the section called "SystemErrors"), vous devriez poursuivre votre examen.

Rubriques

- Examen des erreurs internes du serveur
- Minimisation de l'impact des erreurs internes du serveur
- Amélioration de la conscience opérationnelle

Examen des erreurs internes du serveur

Si vous rencontrez des erreurs internes du serveur dans la table DynamoDB, envisagez les options suivantes :

1. Consultez le tableau de bord AWS Health.

   Pour identifier le problème, la première étape consiste à consulter le tableau de bord de l'intégrité des services AWS et le tableau de bord de l'état du compte AWS. Ces tableaux de bord fournissent des informations précieuses sur les problèmes rencontrés à l'échelle du service, les tables concernées, les problèmes constants et leur cause racine une fois le problème résolu.

   L'examen des informations contenues dans ces tableaux de bord vous permettra de mieux comprendre le statut actuel des Services AWS que vous utilisez et les éventuels problèmes affectant votre compte. Ces informations peuvent vous aider à déterminer les prochaines étapes pour résoudre le problème et minimiser les perturbations de vos opérations.

2. Contactez le Support.

   Si vous observez des erreurs prolongées et persistantes dans vos demandes, cela peut indiquer un problème avec le service. En règle générale, si vous constatez un taux d'échec global de 1 % ou plus au cours des 15 dernières minutes, c'est le moment idéal pour faire remonter le problème à l'équipe d'AWS Support. Consultez le contrat de niveau de service DynamoDB pour en savoir plus.

   Lorsque vous ouvrez un cas auprès de l'équipe d'AWS Support, fournissez les informations suivantes pour accélérer le processus de résolution des problèmes :

   - DDB concerné ; tables ou index secondaires
   - Période pendant laquelle les erreurs ont été observées
   - ID de demande DynamoDB, tels que `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`, disponibles dans les journaux des applications.

   L'inclusion de ces informations dans le cas de support aidera l'équipe AWS à comprendre le problème et à le résoudre plus rapidement. Si vous n'avez pas les ID de demande, vous devez tout de même journaliser le cas avec les autres informations disponibles.

Minimisation de l’impact des erreurs internes du serveur

Si des erreurs internes du serveur se produisent lors de l’utilisation de DynamoDB, minimisez leur impact sur votre application. Prenez en compte les bonnes pratiques suivantes :

- Utilisez les backoffs et les nouvelles tentatives : les comportements par défaut du kit SDK de DynamoDB sont conçus pour trouver le bon équilibre pour la plupart des applications en termes de backoff et de stratégie de nouvelle tentative. Vous pouvez toutefois ajuster ces paramètres en fonction de la tolérance de votre application à la durée d’indisponibilité et de ses exigences de performance. En savoir plus sur les backoffs et les nouvelles tentatives pour comprendre comment optimiser ces paramètres de nouvelle tentative.
- Utilisez des lectures cohérentes à terme : si votre application ne nécessite pas de lectures fortement cohérentes, envisagez d’utiliser des lectures cohérentes à terme. Ces lectures sont moins coûteuses et moins susceptibles de rencontrer des problèmes passagers en raison d’erreurs internes du serveur, car elles seraient effectuées à partir de n’importe quel nœud de stockage disponible. Pour en savoir plus, consultez Cohérence en lecture DynamoDB.

Amélioration de la conscience opérationnelle

Le maintien de la haute disponibilité et de la fiabilité de vos applications est crucial dans le paysage numérique actuel. L’un des principaux aspects de cette approche est la surveillance proactive des erreurs internes du serveur (ISE) dans les tables DynamoDB et les index secondaires globaux (GSI). En créant des alarmes CloudWatch pour surveiller ces erreurs, vous pouvez acquérir une meilleure conscience opérationnelle et être alerté des problèmes potentiels avant qu’ils n’affectent vos utilisateurs finaux. Cette approche est conforme au pilier Excellence opérationnelle du cadre AWS Well-Architected, garantissant ainsi que votre charge de travail DynamoDB est optimisée en termes de performances, de sécurité et de fiabilité.

Création d’alarmes CloudWatch

Vous devez configurer des alarmes CloudWatch sur vos tables DynamoDB pour recevoir des notifications en cas de nombre constamment élevé d’erreurs internes du serveur au lieu d’observer les métriques manuellement. Cela est lié au pilier Excellence opérationnelle du cadre Well-Architected pour toutes les charges de travail sur AWS. Consultez Utilisation du cadre DynamoDB Well-Architected pour optimiser votre charge de travail DynamoDB pour en savoir plus sur la bonne conception d’architecture des tables DynamoDB.

Ces alarmes utilisent des mathématiques de métriques personnalisées pour calculer le pourcentage de demandes ayant échoué pour une fenêtre de 5 minutes. La bonne pratique recommandée consiste à configurer l'alarme pour qu'elle entre dans l'état ALARM lorsque 3 points de données consécutifs dépassent le seuil de 1 %, ce qui signifie qu'au total 1 % des demandes échouent dans un délai de 15 minutes.

L'exemple ci-dessous est un modèle CloudFormation qui peut vous aider à créer des alarmes CloudWatch sur votre table et des GSI sur la table.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Sample template for monitoring DynamoDB
Parameters:
 DynamoDBProvisionedTableName:
    Description: Name of DynamoDB Provisioned Table to create
    Type: String
    MinLength: 3
    MaxLength: 255
    ConstraintDescription : https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/Limits.html#limits-naming-rules
  DynamoDBSNSEmail:
    Description : Email Address subscribed to newly created SNS Topic
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    MinLength: 1
    MaxLength: 255

Resources:
  DynamoDBMonitoringSNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      DisplayName: DynamoDB Monitoring SNS Topic
      Subscription:
        - Endpoint: !Ref DynamoDBSNSEmail
          Protocol: email
      TopicName: dynamodb-monitoring

  DynamoDBTableSystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBTableSystemErrorAlarm'
      AlarmDescription: 'Alarm when system errors exceed 1% of total number of requests
 for 15 minutes'
      AlarmActions:
```

```
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: SystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
```

```
 DynamoDBGSISystemErrorAlarm:
   Type: 'AWS::CloudWatch::Alarm'
   Properties:
     AlarmName: 'DynamoDBGSISystemErrorAlarm'
     AlarmDescription: 'Alarm when GSI system errors exceed 2% of total number of
requests for 15 minutes'
     AlarmActions:
       - !Ref DynamoDBMonitoringSNSTopic
     Metrics:
       - Id: 'e1'
         Expression: 'm1/(m1+m2+m3)'
         Label: GSISystemErrorsOverTotalRequests
       - Id: 'm1'
         MetricStat:
           Metric:
             Namespace: 'AWS/DynamoDB'
             MetricName: 'SystemErrors'
             Dimensions:
               - Name: 'TableName'
                 Value: !Ref DynamoDBProvisionedTableName
               - Name: 'GlobalSecondaryIndexName'
                 Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
           Period: 300
           Stat: 'SampleCount'
           Unit: 'Count'
         ReturnData: False
       - Id: 'm2'
         MetricStat:
           Metric:
             Namespace: 'AWS/DynamoDB'
             MetricName: 'ConsumedReadCapacityUnits'
             Dimensions:
               - Name: 'TableName'
                 Value: !Ref DynamoDBProvisionedTableName
               - Name: 'GlobalSecondaryIndexName'
                 Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
           Period: 300
           Stat: 'SampleCount'
           Unit: 'Count'
         ReturnData: False
       - Id: 'm3'
         MetricStat:
           Metric:
             Namespace: 'AWS/DynamoDB'
```

```
            MetricName: 'ConsumedWriteCapacityUnits'
            Dimensions:
              - Name: 'TableName'
                Value: !Ref DynamoDBProvisionedTableName
              - Name: 'GlobalSecondaryIndexName'
                Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
          Period: 300
          Stat: 'SampleCount'
          Unit: 'Count'
        ReturnData: False
    EvaluationPeriods: 3
    Threshold: 1.0
    ComparisonOperator: 'GreaterThanThreshold'
```

Résolution des problèmes de latence dans Amazon DynamoDB

Si votre charge de travail semble subir une latence élevée, vous pouvez analyser la métrique `SuccessfulRequestLatency` CloudWatch et vérifier la latence moyenne et la latence médiane à l'aide de métriques de percentiles (p50) pour voir si cela est lié à DynamoDB. Une certaine variabilité dans la `SuccessfulRequestLatency` signalée est normale. Les pics occasionnels (en particulier dans la statistique `Maximum` et les percentiles élevés) ne devraient pas être problématiques. Toutefois, si la statistique `Average` ou p50 (médiane) indique une forte augmentation et persiste, vous devez consulter le tableau de bord de l'intégrité des services AWS et le tableau de bord de l'état personnel pour plus d'informations. Parmi les causes possibles, citons la taille de l'élément de la table (un élément de 1 Ko et un élément de 400 Ko varient en matière de latence) ou la taille de la requête (10 éléments contre 100).

Les métriques de percentiles (p99, p90, etc.) peuvent vous aider à mieux comprendre la distribution de la latence. Par exemple :

- p50 (médiane) indique la latence typique de la charge de travail.
- p90 indique que 90 % des demandes sont plus rapides que cette valeur.
- p99 permet d'identifier les pires cas de latence affectant 1 % des demandes.

Des valeurs de p99 élevées avec des valeurs de p50 normales peuvent indiquer des problèmes sporadiques affectant une petite partie des demandes, tandis que des valeurs de p50 constamment élevées peuvent suggérer une certaine dégradation des performances.

> **Note**
>
> Pour analyser des valeurs de percentile personnalisées (telles que p99,9), vous pouvez saisir manuellement le percentile souhaité (par exemple, p99,9) dans le champ de statistique de métrique CloudWatch. Cela vous permet d'évaluer les distributions de latence au-delà des percentiles par défaut répertoriés dans la liste déroulante.

Certaines variations des métriques de latence, en particulier en ce qui concerne les percentiles les plus élevés, sont attendues et peuvent être le résultat d'opérations en arrière-plan pilotées par DynamoDB qui aident à maintenir une disponibilité et une durabilité élevées pour les données stockées dans des tables DynamoDB, ou de problèmes passagers d'infrastructure.

Si nécessaire, envisagez d'ouvrir un dossier de support auprès de AWS Support et continuez à évaluer toutes les options de repli disponibles pour votre application (par exemple, l'évacuation d'une région si vous disposez d'une architecture multirégion) en fonction de vos runbooks. Vous devez journaliser les ID de demande pour les demandes lentes pour la fourniture de ces ID à AWS Support lorsque vous ouvrez un cas de support.

La métrique `SuccessfulRequestLatency` mesure uniquement la latence interne au service DynamoDB. L'activité côté client et les temps de parcours du réseau ne sont pas inclus. Pour en savoir plus sur la latence globale des appels de votre client vers le service DynamoDB, vous pouvez activer la journalisation des métriques de latence dans votre SDK AWS.

> **Note**
>
> Pour la plupart des opérations singleton (opérations qui s'appliquent à un seul élément en spécifiant entièrement la valeur de la clé primaire), DynamoDB fournit `Average SuccessfulRequestLatency` avec une milliseconde à un chiffre. Cette valeur n'inclut pas la surcharge de transport pour le code de l'appelant accédant au point de terminaison DynamoDB. Pour les opérations de données comportant plusieurs éléments, la latence varie en fonction de facteurs tels que la taille du jeu de résultats, la complexité des structures de données renvoyées et les expressions de condition et de filtre appliquées. Pour les opérations multi-éléments répétées sur le même ensemble de données avec les mêmes paramètres, DynamoDB fournit `Average SuccessfulRequestLatency` avec une cohérence élevée.

Envisagez l'une ou plusieurs des stratégies suivantes pour réduire la latence :

- Réutilisez les connexions : les demandes DynamoDB sont effectuées par l'intermédiaire d'une session authentifiée via HTTPS par défaut. L'établissement de la connexion nécessite plusieurs allers-retours et prend du temps, de sorte que la latence de la première demande est plus élevée que celle des demandes suivantes qui réutilisent la connexion. Les demandes effectuées via une connexion déjà initialisée garantissent la faible latence constante de DynamoDB. Pour éviter la surcharge de latence liée à l'établissement de nouvelles connexions, vous pouvez mettre en œuvre un mécanisme « keep-alive » en envoyant une demande `GetItem` toutes les 30 secondes si aucune autre demande n'est faite.
- Utilisez des lectures éventuellement cohérentes : si votre application ne nécessite pas de lectures fortement cohérentes, pensez à utiliser les lectures éventuellement cohérentes par défaut. Les lectures cohérentes à terme ont un coût moins élevé et peuvent provenir de plusieurs zones de disponibilité. Cela permet de sélectionner une zone de disponibilité colocalisée avec le demandeur, ce qui réduit le temps de latence. Pour en savoir plus, consultez Cohérence en lecture DynamoDB.
- Mettez en œuvre la couverture des demandes : pour des exigences de latence très faible de p99, envisagez de mettre en œuvre une couverture des demandes. Avec la couverture des demandes, si la demande initiale ne reçoit pas de réponse assez rapidement, envoyez une deuxième demande équivalente et laissez-les courir, la première réponse l'emporte. Cela améliore la latence de queue au prix de quelques demandes supplémentaires. Vous pouvez décider du temps d'attente avant l'envoi de la deuxième demande. La couverture est plus facile pour les lectures. Pour les écritures, utilisez un ordre basé sur l'horodatage pour garantir que les demandes couvertes sont traitées comme ayant eu lieu au moment de la première tentative, afin d'éviter les mises à jour hors ordre. Cette approche a été décrite dans Timestamp writes for write hedging in Amazon DynamoDB.
- Ajustez le délai d'attente des demandes et le comportement des nouvelles tentatives : le chemin entre le client et DynamoDB passe par de nombreux composants, chacun étant conçu dans une optique de redondance. Tenez compte des aspects suivants :
  - Résilience du réseau
  - Délais d'expiration des paquets TCP
  - Architecture distribuée de DynamoDB

  Les comportements par défaut du kit SDK sont optimisés pour la plupart des applications. Toutefois, vous pouvez mettre en œuvre une stratégie d'interruption immédiate et ajuster les paramètres de délai d'expiration. Les demandes qui prennent beaucoup plus de temps que d'habitude ont moins de chances d'aboutir en fin de compte. En procédant à une interruption

immédiate et à une nouvelle tentative, vous pouvez rapidement réussir en empruntant un chemin différent. Cela équivaut à la couverture des demandes, mais met fin à la première demande au lieu de l'autoriser à se poursuivre.

Évitez de définir des valeurs de délai d'expiration trop basses. Des délais d'expiration trop bas peuvent entraîner des problèmes de disponibilité induits par le client. Par exemple, un délai d'expiration de socket de 50 millisecondes peut provoquer des erreurs de connexion lors des pics de latence du réseau, par exemple à l'approche des limites de bande passante d'instance Amazon EC2 pour le trafic à flux unique. Évaluez soigneusement les avantages de délais d'expiration plus bas par rapport aux risques potentiels pour la disponibilité des applications, et préférez la couverture à des délais d'expiration courts.

Pour une discussion utile sur ce sujet, consultez Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications.

- Réduisez la distance entre le client et le point de terminaison DynamoDB : si vous avez des utilisateurs répartis dans le monde entier, pensez à utiliser Tables globales : réplication multi-active, multirégion. Les tables globales vous permettent de répliquer votre table dans les régions AWS spécifiées dans lesquelles vous voulez qu'elle soit disponible. Vous pouvez placer une copie des données plus près de l'utilisateur final afin de réduire la latence du réseau lors des opérations de lecture et d'écriture. Pour plus d'informations sur l'utilisation efficace des tables globales DynamoDB, consultez Utilisation des tables globales Amazon DynamoDB dans Recommandations AWS.
- Utilisez la mise en cache : si le trafic nécessite beaucoup de lecture, envisagez d'utiliser un des services de mise en cache suivants :
  - DynamoDB Accelerator (DAX) : cache en mémoire entièrement géré et hautement disponible pour DynamoDB, qui offre des performances jusqu'à 10 fois supérieures, de l'ordre de quelques millisecondes à quelques microsecondes, même lorsque le nombre de demandes s'élève à plusieurs millions par seconde. Pour plus d'informations sur DAX, consultez Accélération en mémoire avec DynamoDB Accelerator (DAX) :
  - Amazon ElastiCache : service de cache en mémoire entièrement géré qui peut être intégré à DynamoDB pour améliorer les performances de lecture à l'aide du modèle de cache secondaire. Pour plus d'informations, consultez Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching dans Recommandations AWS.

Résolution des problèmes de limitation dans Amazon DynamoDB

DynamoDB met en œuvre la limitation pour deux objectifs principaux : maintenir les performances globales du service et contrôler les coûts. La limitation sert soit de protection intentionnelle qui empêche la dégradation des performances lorsque les taux de consommation dépassent la capacité, soit de mécanisme de contrôle des coûts lorsque vous atteignez les limites maximales de débit ou de quota de service. En cas de limitation, DynamoDB renvoie des exceptions spécifiques avec des informations détaillées sur les raisons pour lesquelles la demande a été limitée et sur la ressource affectée. Chaque raison de l'étranglement correspond à des CloudWatch indicateurs spécifiques qui fournissent des informations supplémentaires sur la fréquence et les modèles des événements d'étranglement.

Le diagramme suivant illustre les quatre principaux scénarios dans lesquels DynamoDB met en œuvre une limitation protective :

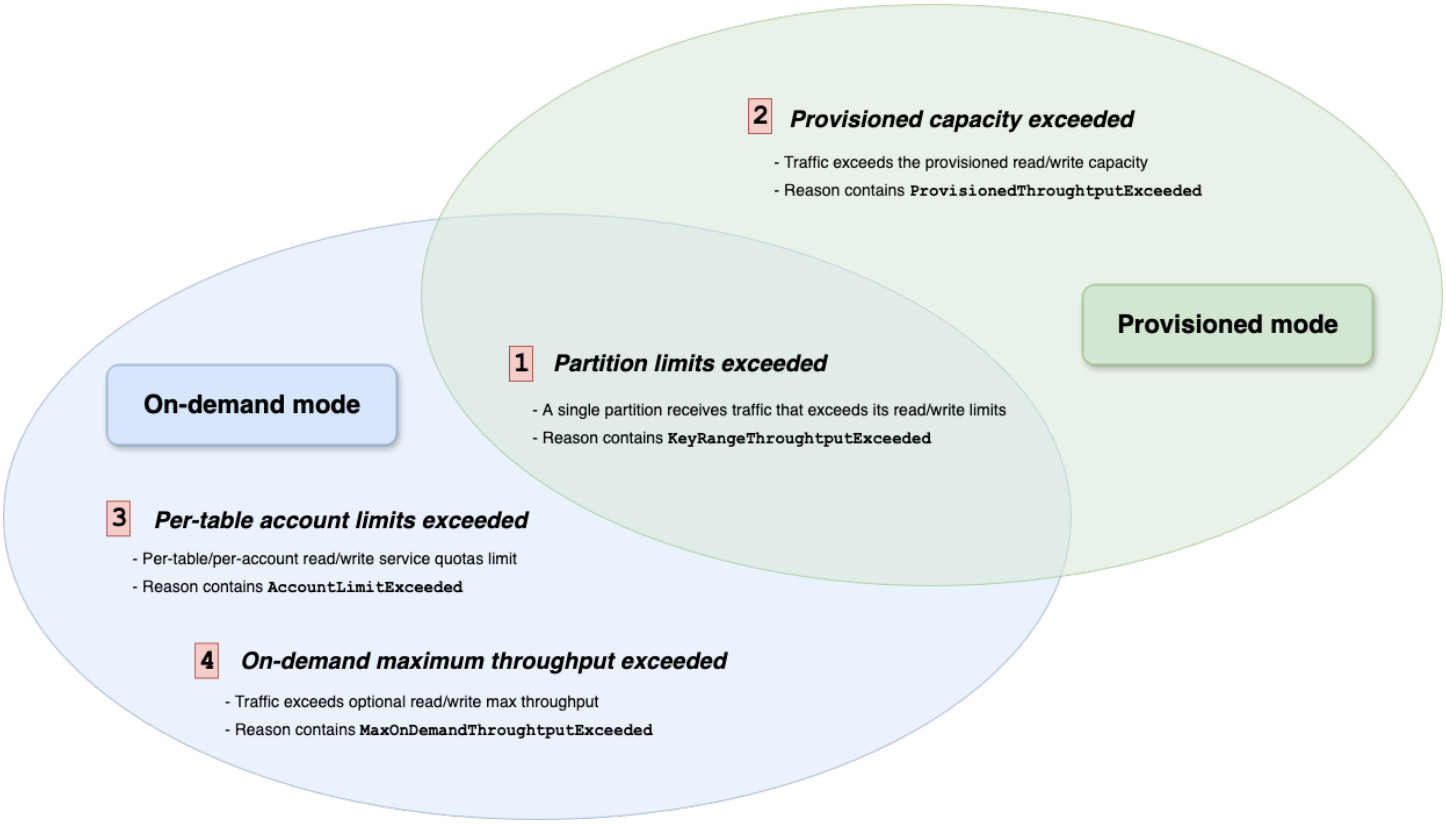
1. Dépassement du débit de plage de clés (dans les deux modes) :

   La consommation dirigée vers des partitions spécifiques dépasse les limites de débit au niveau de la partition interne.
2. Dépassement du débit provisionné (en mode provisionné) :

   La consommation dépasse le nombre d’unités de capacité provisionnée (lecture ou écriture) configurées pour une table ou des index secondaires globaux (GSI).
3. Dépassement des quotas de service au niveau du compte (en mode à la demande) :

   En raison de la consommation, une table ou un GSI dépasse les quotas de service par table au niveau du compte pour le read/write débit dans la région actuelle.AWS Ces quotas servent de garde-fous et peuvent être augmentés.
4. Dépassement du débit maximal à la demande (en mode à la demande) :

   La consommation dépasse les limites de débit maximal configurées pour une table ou des GSI. Il s’agit de limites que vous configurez spécifiquement à des fins de contrôle des coûts.



Ce guide est conçu pour vous aider à comprendre et à utiliser la limitation dans DynamoDB. Tout d'abord, nous vous aidons à identifier le type spécifique de limitation affectant votre charge de travail grâce à un cadre de diagnostic.

Ensuite, la section du guide de résolution propose des conseils spécifiques pour chaque scénario de régulation, notamment des CloudWatch mesures à surveiller à des fins de détection et d'analyse, ainsi que des étapes recommandées pour l'optimisation. En suivant cette approche structurée, vous pouvez mieux diagnostiquer la cause première de la limitation et mettre en œuvre la solution appropriée pour garantir le fonctionnement efficace des tables DynamoDB.

Pour commencer, suivez the section called "Diagnostic des problèmes de limitation" pour savoir comment identifier le type de limitation qui affecte votre charge de travail et comment mettre en œuvre la stratégie de résolution recommandée.

Rubriques

- Diagnostic des problèmes de limitation
- Guide de résolution des problèmes de limitation de DynamoDB
- Compréhension de la limitation d'écriture et de la contre-pression des index secondaires globaux (GSI) dans DynamoDB

- Métriques de limitation CloudWatch

Diagnostic des problèmes de limitation

Lorsque votre application est confrontée à une limitation, DynamoDB fournit des informations détaillées sur les exceptions correspondantes et des métriques CloudWatch ciblées pour vous aider à diagnostiquer ces événements.

Cette section présente une approche systématique pour comprendre les événements de limitation dans vos applications DynamoDB. Elle explique comment interpréter les exceptions de limitation, comment les mettre en corrélation avec les métriques CloudWatch afin d'obtenir des informations plus approfondies et comment déterminer les modifications susceptibles de réduire la limitation dans vos applications DynamoDB.

Comprendre les exceptions de limitation

Lorsque DynamoDB limite une demande, il renvoie des exceptions spécifiques avec des informations de diagnostic détaillées. Parmi ces exceptions, citons notamment `ProvisionedThroughputExceededException`, `RequestLimitExceeded` ou `ThrottlingException`, en Java.

Chaque exception inclut un attribut `ThrottlingReasons`, qui rassemble des éléments `ThrottlingReason` individuels contenant deux champs clés pour vous aider à identifier et à comprendre la limitation :

- Raison : champ concaténé respectant le format `<ResourceType><OperationType><LimitType>`
- ARN de la ressource : Amazon Resource Name (ARN) de la table ou de l'index concerné

Le champ Raison respecte un schéma cohérent qui vous permet de comprendre exactement ce qui se passe :

- ResourceType (ressource affectée par la limitation) : `Table` ou `Index`
- OperationType (type d'opération) : `Read` ou `Write`
- LimitType (raison pour laquelle la limitation a eu lieu) :

- `KeyRangeThroughputExceeded` : cette exception a lieu lorsqu'une partition spécifique sur laquelle repose votre table ou votre index a consommé une capacité de lecture ou d'écriture dépassant les limites de débit internes par partition.
- `ProvisionedThroughputExceeded` : cette exception a lieu sur une table provisionnée ou un index secondaire global provisionné lorsque le taux de consommation en lecture ou en écriture a dépassé le montant provisionné.
- `AccountLimitExceeded` : cette exception a lieu sur une table à la demande ou un index à la demande lorsque le taux de consommation en lecture ou en écriture a dépassé le taux de consommation maximal pour une table et ses index, tel que défini au niveau du compte. Vous pouvez demander une augmentation de ce quota.
- `MaxOnDemandThroughputExceeded` : cette exception a lieu sur une table à la demande ou un index à la demande lorsque le taux de consommation en lecture ou en écriture a dépassé le taux de consommation maximal fourni par l'utilisateur et configuré pour cette table ou cet index. Vous pouvez vous-même augmenter cette valeur à votre gré dans les limites spécifiées au niveau du compte ou la définir sur -1 pour indiquer qu'aucune limite n'est fournie par l'utilisateur.

L'ARN de la ressource identifie précisément la table ou l'index qui est limité :

- Pour les tables : `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`
- Pour les index : `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`

Exemples de raisons complètes de limitation :

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

Cela permet d'identifier exactement quelle ressource est limitée, quel type d'opération en est la cause et pourquoi cette limitation a eu lieu.

Exemples d'exception

Exemple 1 : dépassement de la capacité provisionnée sur un GSI

```
{
    "ThrottlingReasons": [
```

```
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
        }
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

Dans cet exemple, l'application reçoit une exception `ProvisionedThroughputExceededException` avec la raison `IndexWriteProvisionedThroughputExceeded`. Les écritures dans l'index `OrderDateIndex` sont limitées, car la consommation en écriture a dépassé la capacité d'écriture provisionnée configurée pour ce GSI.

Exemple 2 : dépassement du débit maximal à la demande

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
```

```
}
```

Dans cet exemple, les lectures de la table `UserSessions` sont limitées, car elles dépassent la limite de débit maximale à la demande configurée pour cette table.

Cadre de diagnostic des limitations DynamoDB

Lorsque votre application est confrontée à une limitation, procédez comme suit pour diagnostiquer et résoudre le problème.

Étape 1 : analyser les détails du champ **`ThrottlingReason`**

1. Vérifiez le champ Raison pour identifier la raison précise de la limitation. Celui-ci détaille le type de ressource limitée (table ou index), le type d'opération à l'origine de l'événement de limitation (lecture ou écriture) et le type de limite dépassé (partition, débit provisionné, limite de compte).
2. Vérifiez le champ resourceArn pour identifier la ressource (table ou GSI) qui est limitée.
3. Utilisez ces informations combinées pour comprendre le contexte complet du problème de limitation.

   Par exemple, imaginez un scénario dans lequel vous recevez l'exception `ProvisionedThroughputExceededException` suivante avec la raison de limitation `TableWriteKeyRangeThroughputExceeded`. Le champ resourceArn concerné est `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   Cette combinaison vous indique que les opérations d'écriture dans votre table `CustomerOrders` sont limitées. La limitation a lieu au niveau de la partition (et non au niveau de la table, auquel cas `TableWriteProvisionedThroughputExceeded` serait affiché). La cause racine est que vous avez dépassé la capacité de débit maximale pour une valeur ou une plage de clés de partition spécifique, ce qui indique un problème de partition critique.

   Comprendre cette relation entre les différents éléments de l'exception vous aide à mettre en œuvre la stratégie d'atténuation appropriée. Dans ce cas, il s'agit de traiter la partition critique plutôt que d'augmenter la capacité provisionnée globale de la table.

Étape 2 : identifier et analyser les métriques CloudWatch associées

1. Identifiez vos métriques : chaque raison de limitation dans DynamoDB correspond directement à des métriques CloudWatch spécifiques que vous pouvez surveiller pour suivre et analyser

les événements de limitation. Vous pouvez systématiquement déduire les noms de métriques CloudWatch appropriés à partir de la raison de la limitation.

2. Pour associer la raison de la limitation aux métriques CloudWatch correspondantes, reportez-vous à ce tableau de référence :

   Raisons complètes de la limitation et référence des métriques CloudWatch

Catégorie	Raison de la limitation	Métriques CloudWatch principales
Dépassement de la capacité provisionnée	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Dépassement des limites de partition	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)

Catégorie	Raison de la limitation	Métriques CloudWatch principales
Dépassement du débit maximal à la demande	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Dépassement des limites du compte	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSI)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSI)

Par exemple, si vous avez reçu `IndexWriteProvisionedThroughputExceeded`, vous devriez au minimum surveiller la métrique `WriteProvisionedThroughputThrottleEvents` de CloudWatch pour l'index spécifique identifié dans le champ `ResourceArn`.

3. Surveillez ces métriques dans CloudWatch pour comprendre la fréquence et le timing des événements de limitation, pour différencier la limitation en lecture de la limitation en écriture, pour identifier les modèles temporels en cas d'augmentation de la limitation et pour suivre les tendances en matière d'utilisation de la capacité.

   DynamoDB publie des métriques détaillées pour chaque table et chaque index secondaire global. Les métriques (`ReadThrottleEvents`, `WriteThrottleEvents` et `ThrottledRequests`) regroupent tous les événements de limitation de votre table et de ses index.

Étape 3 : identifier les clés limitées et les taux d'accès élevés à l'aide de CloudWatch Contributor Insights (pour la limitation liée aux partitions)

Si vous avez identifié des problèmes liés aux partitions à l'étape 1 (tels que des erreurs `KeyRangeThroughputExceeded`), CloudWatch Contributor Insights pour DynamoDB peut vous aider à identifier les clés spécifiques qui génèrent votre trafic et qui sont confrontées à des événements de limitation dans votre table ou votre index.

1. Activez CloudWatch Contributor Insights pour votre table ou index limité en fonction de votre `ResourceARN`.

   Vous pouvez choisir le mode Clés limitées pour vous concentrer exclusivement sur les clés les plus limitées. Ce mode est idéal pour la surveillance continue, car il ne traite les événements qu'en cas de limitation. Sinon, le mode Clés consultées et limitées vous permet d'identifier des modèles récurrents au niveau des clés les plus consultées.
2. Analysez les rapports pour identifier les modèles problématiques. Recherchez les clés présentant des taux d'accès ou de limitation excessivement élevés, puis mettez en corrélation la limitation avec les modèles de trafic. Vous pouvez créer des tableaux de bord intégrés combinant les graphiques de Contributor Insights et les métriques de DynamoDB CloudWatch.

Pour obtenir des informations détaillées sur l'activation et l'utilisation de CloudWatch Contributor Insights, consultez Using CloudWatch Contributor Insights for DynamoDB.

Étape 4 : déterminer la solution appropriée

Après avoir diagnostiqué la cause spécifique de la limitation, mettez en œuvre la solution recommandée en fonction de votre contexte spécifique. La solution appropriée dépend de plusieurs facteurs, notamment de votre scénario de limitation, du mode de capacité de la table, des décisions de conception de la table et des clés, de l'efficacité des requêtes et des modèles d'accès, de la

configuration des index globaux et secondaires, ainsi que de l'architecture globale du système et des points d'intégration.

Pour obtenir des solutions détaillées répondant à vos scénarios de limitation spécifiques, consultez la section the section called "Guide de résolution". Cette ressource fournit des stratégies de correction ciblées adaptées à la raison particulière de la limitation et à la configuration du mode de capacité.

Étape 5 : suivre vos progrès

1. Suivez les métriques CloudWatch qui correspondent à votre scénario de limitation.
2. Pour confirmer que vos stratégies d'atténuation sont efficaces, vous devriez constater une diminution des événements de limitation.

Guide de résolution des problèmes de limitation de DynamoDB

Cette section fournit des conseils de résolution ciblés pour chaque raison de limitation spécifique que DynamoDB peut renvoyer. Chaque entrée inclut des étapes de résolution suggérées basées sur les bonnes pratiques et les métriques CloudWatch correspondantes à surveiller.

DynamoDB implémente 16 raisons de limitation distinctes réparties dans quatre catégories principales. Utilisez les raisons de limitation indiquées dans l'exception de votre application pour accéder directement aux instructions pertinentes.

Dépassement du débit de la plage de clés (partitions critiques)

Ces raisons de limitation se produisent lorsque des partitions individuelles dépassent leurs limites de débit, ce qui affecte à la fois le mode provisionné et le mode à la demande :

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Dépassement du débit provisionné

Ces raisons de limitation surviennent lorsque les taux de consommation dépassent les limites de capacité provisionnée en mode provisionné :

- TableReadProvisionedThroughputExceeded

- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Dépassement des limites de compte

Ces raisons de limitation se produisent lorsque les taux de consommation dépassent les quotas de débit au niveau du compte dans votre région AWS :

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

Dépassement du débit maximal à la demande

Ces raisons de limitation se produisent lorsque les taux de consommation dépassent les limites de débit maximales configurées en mode à la demande :

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1- Dépassement du débit de la plage de clés (partitions critiques)

Amazon DynamoDB impose des limites de débit spécifiques au niveau de la partition pour les tables et les index secondaires globaux (GSI). Chaque partition possède un nombre maximum d'unités de capacité de lecture (RCUs) et d'unités de capacité d'écriture (WCUs) par seconde. Lorsque les partitions reçoivent un trafic concentré qui dépasse ces limites, elles sont soumises à une limitation tandis que les autres opérations peuvent rester sous-utilisées, ce qui crée des « partitions critiques ». La limitation au niveau des partitions de DynamoDB fonctionne indépendamment pour les lectures et les écritures : une partition peut limiter les lectures alors que les écritures se poursuivent normalement, ou vice versa. Cette limitation peut se produire même lorsque votre table ou votre GSI dispose d’une capacité globale suffisante. Ressources supplémentaires :

- Pour en savoir plus sur les limites de partition DynamoDB et sur la conception efficace des clés de partition pour prévenir les partitions critiques, consultez Bonnes pratiques pour la conception et l'utilisation performantes de clés de partition dans DynamoDB.
- Pour en savoir plus sur les concepts généraux de partition et sur la distribution des données, consultez Partitions in DynamoDB.
- Pour obtenir des conseils supplémentaires et des scénarios concrets permettant de gérer les clés de partition et le débit, consultez the section called "Ressources supplémentaires".

Lorsque des partitions individuelles dépassent leurs limites de débit, DynamoDB renvoie le type de raison de limitation `KeyRangeThroughputExceeded` dans l'exception correspondante. Les informations indiquent qu'une partition connaît un trafic élevé et précisent quel type d'opération (lecture ou écriture) est à l'origine du problème.

Mesures d'atténuation du dépassement du débit de la plage de clés

Cette section fournit des conseils de résolution pour les scénarios de limitation au niveau des partitions. Avant d'utiliser ce guide, assurez-vous d'avoir identifié les raisons spécifiques de la limitation liées à la gestion des exceptions par votre application et d'avoir déterminé l'Amazon Resource Name (ARN) de la ressource concernée. Pour en savoir plus sur la façon de déterminer les raisons de la limitation et d'identifier des ressources limitées, consultez the section called "Cadre de diagnostic".

Avant de vous lancer dans des scénarios de limitation spécifiques, vérifiez si le problème se résout automatiquement :

- DynamoDB s'adapte souvent aux partitions chaudes grâce à son mécanisme automatique. split-for-heat Si vous constatez des événements de limitation qui s'arrêtent après un court laps de temps, votre table s'est peut-être déjà adaptée en fragmentant la partition critique. Lorsque les partitions sont fragmentées, chaque nouvelle partition gère une plus petite partie de l'espace de clés, ce qui permet de répartir la charge de manière plus uniforme. Dans de nombreux cas, aucune autre action n'est nécessaire, car DynamoDB a automatiquement résolu le problème.

  Pour plus d'informations sur le split-for-heatmécanisme, consultezthe section called "Ressources supplémentaires".

Si la limitation persiste, reportez-vous aux scénarios spécifiques ci-dessous pour connaître les options de correction ciblées :

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

Quand cela se produit

Le taux de consommation d’une ou de plusieurs partitions de votre table DynamoDB dépasse la limite de débit de lecture de la partition. Cette limitation a lieu quelle que soit la capacité totale allouée de votre table et affecte à la fois les tables provisionnées et à la demande. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

Pour le mode provisionné et le mode à la demande :

- Capacité de préchauffage : si la limitation persiste, vérifiez si votre table est limitée par sa capacité the section called “Débit chaud”. Utilisez un débit à chaud ou augmentez la capacité de lecture provisionnée à l’avance pour faire face aux augmentations de trafic attendues. L’augmentation du débit à chaud améliore la capacité de la table à gérer les pics de trafic soudains avant que la limitation ne se produise. Au fil du temps, si le débit réel se rapproche régulièrement des niveaux de débit critiques, DynamoDB peut diviser les partitions occupées en fonction des modèles d’utilisation observés.
- Identifiez les clés critiques : si la table ne l’a pas résolu automatiquement et que le débit à chaud est élevé ou si l’augmentation du débit n’a pas aidé, vous devez identifier les clés critiques spécifiques. Utilisez the section called “Identifier les touches de raccourci à l'aide de CloudWatch Contributor Insights” pour déterminer si certaines valeurs de clé de partition sont critiques. Il s’agit de la première étape à suivre pour cibler efficacement les efforts d’atténuation. Notez que l’identification n’est pas toujours simple, en particulier dans le cas de partitions dynamiques (où différentes partitions deviennent critiques au fil du temps) ou lorsque la limitation est déclenchée par des opérations telles que des analyses. Pour ces scénarios complexes, vous devrez peut-être analyser les modèles d’accès de votre application et les mettre en corrélation avec le moment où les événements de limitation ont eu lieu.

- En fonction de votre cas d'utilisation, envisagez d'utiliser des lectures à terme cohérentes : passez d'une lecture très cohérente à une lecture finalement cohérente, qui consomme la moitié de votre capacité de lecture effective RCUs et peut immédiatement doubler votre capacité de lecture effective. Pour connaître les bonnes pratiques relatives à la mise en œuvre de la lecture cohérente à terme afin de réduire la consommation de la capacité de lecture, consultez the section called “Cohérence en lecture DynamoDB”.
- Améliorez la conception des clés de partition : comme solution à long terme, envisagez the section called “Amélioration de la conception des clés de partition” pour répartir l’accès de manière plus uniforme entre les partitions. Cette approche apporte souvent la solution la plus complète aux problèmes de partition critique en s’attaquant à la cause racine. Cependant, elle nécessite une planification minutieuse, car elle implique d’importants défis en matière de migration.

TableWriteKeyRangeThroughputExceeded

Quand cela se produit

Le taux de consommation d’une ou de plusieurs partitions de votre table DynamoDB dépasse la limite de débit d’écriture de la partition. Cette limitation a lieu quelle que soit la capacité totale allouée de votre table et affecte à la fois les tables provisionnées et à la demande. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

Pour le mode provisionné et le mode à la demande :

- Capacité de préchauffage : si la limitation persiste, vérifiez si votre table est limitée par sa capacité Présentation du débit chaud DynamoDB. Utilisez un débit à chaud ou augmentez la capacité d’écriture provisionnée à l’avance pour faire face aux augmentations de trafic attendues. L’augmentation du débit à chaud améliore la capacité de la table à gérer les pics de trafic soudains avant que la limitation ne se produise. Au fil du temps, si le débit réel se rapproche régulièrement des niveaux de débit critiques, DynamoDB peut diviser les partitions occupées en fonction des modèles d’utilisation observés.
- Identifiez les clés critiques : si la table ne l’a pas résolu automatiquement et que le débit à chaud est élevé ou si l’augmentation du débit n’a pas aidé, vous devez identifier les clés critiques spécifiques. Utilisez the section called “Identifier les touches de raccourci à l'aide de CloudWatch

Contributor Insights" pour déterminer si certaines valeurs de clé de partition sont critiques. Il s'agit de la première étape à suivre pour cibler efficacement les efforts d'atténuation. Tenez compte de ces modèles courants :

- Si la même clé de partition apparaît fréquemment dans vos données de limitation, cela indique une clé critique concentrée.
- Si vous ne voyez pas de clés répétées mais que vous écrivez des données de manière ordonnée (par exemple, des horodatages séquentiels ou des opérations basées sur une analyse, qui suivent l'ordre de l'espace de clés), vous avez probablement des partitions critiques dynamiques où les différentes clés deviendront critiques au fil du temps lorsque vos écritures passeront par l'espace de clés.

Notez que la limitation de l'écriture peut également se produire avec des opérations telles que `BatchWriteItem` ou avec des transactions qui affectent plusieurs éléments simultanément. Lorsque des éléments individuels d'une demande `BatchWriteItem` sont limités, DynamoDB ne propage pas ces erreurs de limitation dans le code de l'application. DynamoDB renvoie plutôt des informations sur les éléments non traités dans la réponse. Votre application devra les gérer en réessayant de traiter ces éléments spécifiques. Pour les transactions, l'ensemble de l'opération échoue avec une exception `TransactionCanceledException` si un élément quelconque est limité. Pour ces scénarios complexes, vous devrez peut-être analyser les modèles d'écriture et les flux de travail d'ingestion de données de votre application, les mettre en corrélation avec le moment où les événements de limitation ont lieu et mettre en œuvre des stratégies appropriées de gestion des nouvelles tentatives.

- Améliorez la conception des clés de partition : comme solution à long terme, envisagez the section called "Amélioration de la conception des clés de partition" pour répartir l'accès de manière plus uniforme entre les partitions. Cette approche apporte souvent la solution la plus complète aux problèmes de partition critique en s'attaquant à la cause racine. Cependant, elle nécessite une planification minutieuse, car elle implique d'importants défis en matière de migration.

IndexReadKeyRangeThroughputExceeded

Quand cela se produit

Le taux de consommation d'une ou de plusieurs partitions de votre GSI DynamoDB dépasse la limite de débit de lecture de la partition. Cette limitation a lieu quelle que soit la capacité totale allouée de votre GSI et affecte à la fois les tables provisionnées et à la demande. Vous pouvez surveiller les CloudWatch métriques the section called "Techniques courantes de diagnostic et de surveillance" pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Capacité de préchauffage : si la limitation persiste, vérifiez si votre GSI est limité par sa capacité Présentation du débit chaud DynamoDB. Utilisez un débit à chaud ou augmentez la capacité de lecture provisionnée à l'avance pour faire face aux augmentations de trafic attendues. L'augmentation du débit à chaud améliore la capacité du GSI à gérer les pics de trafic soudains avant que la limitation ne se produise. Au fil du temps, si le débit réel se rapproche régulièrement des niveaux de débit critiques, DynamoDB peut diviser les partitions occupées en fonction des modèles d'utilisation observés.
- Identifiez les clés critiques : si le GSI ne l'a pas résolu automatiquement et que le débit à chaud est élevé ou si l'augmentation du débit n'a pas aidé, vous devez identifier les clés critiques spécifiques. Utilisez the section called "Identifier les touches de raccourci à l'aide de CloudWatch Contributor Insights" pour déterminer si certaines valeurs de clé de partition sont critiques. Il s'agit de la première étape à suivre pour cibler efficacement les efforts d'atténuation. Notez que pour GSIs, la distribution des clés de partition peut différer considérablement de celle de votre table de base, ce qui crée différents modèles de touches de raccourci.
- Modifiez la conception des clés de partition GSI : déterminez si la conception des clés de votre GSI ne crée pas de points critiques artificiels (tels que des indicateurs d'état, des clés de date uniquement ou des attributs booléens) qui concentrent les lectures sur un petit nombre de partitions. Envisagez d'utiliser des clés composites qui combinent l'attribut de faible cardinalité avec un attribut de cardinalité élevée (par exemple, ACTIVE#customer123 au lieu de simplement ACTIVE) ou d'appliquer des techniques the section called "Partitionnement d'écriture" aux éléments de la table de base qui affectent la distribution des GSI afin de répartir les écritures entre plusieurs partitions. Bien que l'interrogation de données fragmentées nécessite une logique d'application supplémentaire pour agréger les résultats, cette approche empêche la limitation en répartissant les modèles d'accès de manière plus uniforme.

IndexWriteKeyRangeThroughputExceeded

Quand cela se produit

Le taux de consommation d'une ou de plusieurs partitions de votre GSI DynamoDB dépasse la limite de débit d'écriture de la partition. Cette limitation a lieu quelle que soit la capacité totale allouée de votre GSI et affecte à la fois les tables provisionnées et à la demande. Vous pouvez surveiller les

CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Modifiez les clé de partition des GSI : passez en revue la conception des clés de partition de votre GSI pour vérifier qu’elle a une cardinalité (unicité) suffisante pour répartir les écritures de manière uniforme. L’une des causes fréquentes de la limitation de l’écriture dans un GSI est l’utilisation d’attributs à faible cardinalité comme clés de partition du GSI (tels que des indicateurs d’état avec seulement quelques valeurs possibles). Même lorsque votre table de base possède des clés de partition bien distribuées, votre GSI peut rencontrer des partitions critiques si sa clé de partition concentre les écritures sur un petit nombre de valeurs. Par exemple, si 80 % de vos éléments ont le statut actif, cela crée une partition critique sévère dans un GSI basé sur le statut. Envisagez d’utiliser des clés composites qui combinent l’attribut de faible cardinalité avec un attribut de cardinalité élevée (par exemple, ACTIVE#customer123 au lieu de simplement ACTIVE) ou d’appliquer des techniques the section called “Partitionnement d’écriture” aux éléments de la table de base qui affectent la distribution des GSI afin de répartir les écritures entre plusieurs partitions. Bien que l’interrogation de données fragmentées nécessite une logique d’application supplémentaire pour agréger les résultats, cette approche empêche la limitation en répartissant les modèles d’accès de manière plus uniforme.
- Capacité de préchauffage : vérifiez si votre GSI est limité par sa capacité Présentation du débit chaud DynamoDB. Utilisez un débit à chaud ou augmentez la capacité de lecture provisionnée à l’avance pour faire face aux augmentations de trafic attendues. L’augmentation du débit à chaud améliore la capacité du GSI à gérer les pics de trafic soudains avant que la limitation ne se produise. Au fil du temps, si le débit réel se rapproche régulièrement des niveaux de débit critiques, DynamoDB peut diviser les partitions occupées en fonction des modèles d’utilisation observés.
- Optimisez les projections GSI : appliquez the section called “Optimisation des projections des GSI” des techniques pour réduire le volume d'écriture à GSIs. La projection d’un nombre réduit d’attributs peut limiter considérablement la capacité d’écriture consommée par chaque mise à jour du GSI.

Techniques courantes de diagnostic et de surveillance

Lors du dépannage de la régulation au niveau des partitions, plusieurs CloudWatch indicateurs peuvent aider à identifier les partitions chaudes et à en confirmer la cause première.

CloudWatch Indicateurs essentiels

Surveillez ces métriques clés pour diagnostiquer la limitation au niveau des partitions :

- Événements de limitation au niveau des partitions : `ReadKeyRangeThroughputThrottleEvents` et `WriteKeyRangeThroughputThrottleEvents` suivent les partitions individuelles qui dépassent leurs limites de débit. `ReadThrottleEvents` et `WriteThrottleEvents` suivent les demandes de lecture ou d'écriture qui dépassent la capacité allouée.
- Consommation de capacité : `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits` affichent les modèles d'utilisation généraux.

Procédures de résolution

Identifier les touches de raccourci à l'aide de CloudWatch Contributor Insights

Utilisez cette procédure pour identifier les clés de partition qui sont à l'origine de la limitation.

1. Activez CloudWatch Contributor Insights sur votre table ou GSI pour suivre les touches les plus limitées. Envisagez de CloudWatch laisser Contributor Insights activé en permanence pour les alertes de régulation en temps réel en utilisant le mode touches limitées. Ce mode se concentre exclusivement sur les demandes limitées en traitant uniquement les événements lorsque la limitation se produit. Cette surveillance ciblée est un moyen rentable d'avoir une visibilité continue sur les problèmes de limitation.
2. Identifiez les clés qui sont à l'origine des problèmes de partition critique.
3. (Si le mode clés consultées et limitées est activé) Analysez les modèles d'accès au fil du temps pour déterminer si les clés critiques sont constantes ou apparaissent pendant des périodes spécifiques.

Amélioration de la conception des clés de partition

Utilisez cette approche lorsque vous pouvez modifier le schéma de votre table afin de mieux répartir le trafic entre les partitions. Lorsque cela est possible, il s'agit de la solution à long terme la plus efficace pour les problèmes de partition critique. Idéalement, la conception des clés de partition doit être examinée de près lors de la phase initiale de conception de la table.

La modification de la conception des clés de partition représente une altération fondamentale de votre modèle de données, qui a un impact sur l'ensemble de votre écosystème d'applications. Avant de procéder avec cette approche, prenez connaissance de ces limites majeures :

- Complexité de la migration des données : la modification de la conception des clés de partition nécessite la migration de toutes les données existantes, ce qui peut s'avérer fastidieux en termes de ressources et de temps pour les grandes tables.
- Modifications du code de l'application : tout le code de l'application qui lit ou écrit dans la table doit être mis à jour afin de pouvoir utiliser la nouvelle structure de clés.
- Impact sur la production : la migration vers une nouvelle conception de clés implique souvent une durée d'indisponibilité ou des stratégies complexes de double écriture pendant la transition.

Pour obtenir des conseils et des principes complets sur la conception des clés de partition, consultez the section called "Création de clés de partition" et the section called "Répartition des charges de travail".

Optimisation des projections des GSI

Passez en revue les modèles de requête de votre application pour déterminer exactement quels attributs doivent être disponibles lorsque vous interrogez le GSI, et limitez vos projections à ces seuls attributs. Lorsque vous mettez à jour des attributs qui ne sont pas projetés dans un GSI, aucune opération d'écriture n'est effectuée sur ce GSI, ce qui réduit la consommation du débit d'écriture pendant les mises à jour. Cette stratégie de projection ciblée optimise à la fois les performances et les coûts tout en répondant aux exigences de votre application en matière de requêtes. Notez que la projection d'un nombre réduit d'attributs limite la consommation de la capacité d'écriture, mais peut nécessiter des lectures supplémentaires de la table de base.

Pour plus d'informations sur les stratégies de projection efficaces, consultez Best Practices for Using Secondary Indexes in DynamoDB.

Ressources supplémentaires

Les articles de blog suivants fournissent des exemples concrets et des détails pratiques sur les concepts abordés dans ce guide :

- Pour plus d'informations sur l'utilisation GSIs pour distribuer le trafic de lecture, consultez la section Utilisation d'index secondaires globaux pour créer des index secondaires cohérents dans Amazon DynamoDB.

- Pour des conseils pratiques sur la mise à l'échelle de DynamoDB et sur la gestion des partitions critiques, consultez Part 1: Scaling DynamoDB - How partitions, hot keys, and split for heat impact performance.
- Pour des informations détaillées sur le fonctionnement du split-for-heat mécanisme de DynamoDB, ses avantages et les détails de mise en œuvre, consultez la partie 3 : Résumé et meilleures pratiques.
- Pour des stratégies détaillées de partitionnement d'écriture, consultez the section called "Partitionnement d'écriture".

2- Dépassement du débit provisionné

La limitation de capacité allouée se produit lorsque le taux de consommation de votre application dépasse les unités de capacité de lecture ou d'écriture (RCUs/WCUs) configurées pour vos tables ou vos index secondaires globaux. Alors que DynamoDB fournit une capacité de débordement pour gérer les pics de trafic occasionnels, les demandes soutenues dépassant les limites que vous avez allouées entraînent une limitation. Dans ce cas, DynamoDB renvoie le type de raison de limitation `ProvisionedThroughputExceeded` dans l'exception de limitation. La raison indique si le problème concerne les opérations de lecture ou d'écriture et s'il affecte la table de base ou un index secondaire global.

La limitation peut se produire, qu'Auto Scaling soit activé ou non. Auto Scaling s'adapte à l'augmentation de la consommation, mais ne réagit pas instantanément et est soumis aux limites de capacité maximale que vous configurez. En d'autres termes, la limitation peut toujours se produire lors de pics de trafic soudains ou lorsque la consommation dépasse vos limites Auto Scaling maximales.

Le débit provisionné a dépassé les mesures d'atténuation

Cette section fournit des conseils de résolution pour les scénarios de limitation de la capacité provisionnée. Avant d'utiliser ce guide, assurez-vous d'avoir identifié la raison spécifique de la limitation liée à la gestion des exceptions par votre application et d'avoir déterminé l'Amazon Resource Name (ARN) de la ressource concernée. Pour en savoir plus sur la façon de déterminer les raisons de la limitation et d'identifier des ressources limitées, consultez the section called "Cadre de diagnostic".

Avant de vous plonger dans des scénarios de limitation spécifiques, déterminez si la limitation est réellement un problème à résoudre :

- Des limitations occasionnelles sont normales et attendues dans les applications DynamoDB bien optimisées. Une limitation indique simplement que vous consommez 100 % de ce que vous avez provisionné. Si votre application gère correctement la limitation lors des nouvelles tentatives et que vos performances globales répondent aux exigences, il est possible que la limitation ne nécessite pas d'action immédiate.
- Toutefois, si la limitation entraîne une latence inacceptable côté client, détériore l'expérience utilisateur ou empêche les opérations critiques de se terminer en temps voulu, appliquez les mesures d'atténuation ci-dessous.

Lorsque vous devez résoudre des problèmes de limitation, déterminez d'abord si la limitation est causée par :

- Des pics de trafic temporaires : augmentations de trafic de courte durée qui dépassent votre capacité provisionnée, mais qui ne sont pas durables. Ces pics de trafic nécessitent des stratégies différentes de celles d'un trafic élevé continu.
- Un trafic élevé continu : charges de travail soutenues qui dépassent constamment la capacité provisionnée.

Pour les pics de trafic, envisagez de recourir aux stratégies décrites dans le blog Gérer les pics de trafic avec la capacité provisionnée d'Amazon DynamoDB dans the section called "Ressources supplémentaires".

Pour un trafic élevé continu, envisagez de recourir aux options d'ajustement de la capacité ci-dessous :

- the section called "TableReadProvisionedThroughputExceeded"
- the section called "TableWriteProvisionedThroughputExceeded"
- the section called "IndexReadProvisionedThroughputExceeded"
- the section called "IndexWriteProvisionedThroughputExceeded"

TableReadProvisionedThroughputExceeded

Quand cela se produit

Le taux de consommation de lecture de votre application dépasse les unités de capacité de lecture allouées (RCUs) configurées pour votre table. Vous pouvez surveiller les CloudWatch métriques

the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Envisagez les stratégies suivantes pour résoudre le problème de limitation de la capacité de lecture :

- Passez en mode de capacité à la demande : envisagez de faire passer votre table en mode à la demande si vous êtes fréquemment confronté à une limitation due à des pics de trafic. Le mode à la demande élimine les problèmes de provisionnement et adapte automatiquement la capacité à votre charge de travail.
- Si vous restez en mode provisionné et qu’Auto Scaling n’est pas activé :
  - Envisagez d’augmenter la capacité de lecture de la table.
  - Activez Auto Scaling pour la capacité de lecture au niveau de votre table.
- Si Auto Scaling est activé (par défaut pour les tables créées dans la console) :
  - Optimisez les paramètres Auto Scaling de lecture de votre table.

TableWriteProvisionedThroughputExceeded

Quand cela se produit

Le taux de consommation d'écriture de votre application dépasse les unités de capacité d'écriture allouées (WCUs) configurées pour votre table. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Envisagez les stratégies suivantes pour résoudre le problème de limitation de la capacité d’écriture :

- Passez en mode de capacité à la demande : envisagez de faire passer votre table en mode à la demande si vous êtes fréquemment confronté à une limitation due à des pics de trafic. Le mode à la demande élimine les problèmes de provisionnement et adapte automatiquement la capacité à votre charge de travail.
- Si vous restez en mode provisionné et qu’Auto Scaling n’est pas activé :
  - Envisagez d’augmenter la capacité d’écriture de la table.
  - Activez Auto Scaling pour la capacité d’écriture au niveau de votre table.

- Si Auto Scaling est activé (par défaut pour les tables créées dans la console) :
  - Optimisez les paramètres Auto Scaling d'écriture dans votre table.

IndexReadProvisionedThroughputExceeded

Quand cela se produit

La consommation de lecture sur un index secondaire global (GSI) dépasse les unités de capacité de lecture allouées au GSI (). RCUs Vous pouvez surveiller les CloudWatch métriques the section called "Techniques courantes de diagnostic et de surveillance" pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Envisagez les stratégies suivantes pour résoudre le problème de limitation de la capacité de lecture du GSI :

- Passez en mode de capacité à la demande : envisagez de faire passer la table de base en mode à la demande si vous êtes fréquemment confronté à une limitation due à des pics de trafic. Le mode à la demande élimine les problèmes de provisionnement et adapte automatiquement la capacité à votre charge de travail.
- Si vous restez en mode provisionné et qu'Auto Scaling n'est pas activé :
  - Envisagez d'augmenter la capacité de lecture du GSI.
  - Activez Auto Scaling pour la capacité de lecture au niveau de votre GSI.
- Si Auto Scaling est activé (par défaut pour les tables créées dans la console) :
  - Optimisez les paramètres Auto Scaling de lecture de votre GSI.

IndexWriteProvisionedThroughputExceeded

Quand cela se produit

Les mises à jour des éléments de la table de base déclenchent des écritures dans un GSI qui dépassent la capacité d'écriture provisionnée du GSI. Cela entraîne une contre-pression de limitation au niveau des écritures de la table de base. Vous pouvez surveiller les CloudWatch métriques the section called "Techniques courantes de diagnostic et de surveillance" pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Envisagez les stratégies suivantes pour résoudre le problème de limitation de la capacité d'écriture du GSI :

- Passez en mode de capacité à la demande : envisagez de faire passer la table de base en mode à la demande si vous êtes fréquemment confronté à une limitation due à des pics de trafic. Le mode à la demande élimine les problèmes de provisionnement et adapte automatiquement la capacité à votre charge de travail.
- Si vous restez en mode provisionné et qu'Auto Scaling n'est pas activé :
  - Envisagez d'augmenter la capacité d'écriture du GSI.
  - Activez Auto Scaling pour la capacité d'écriture au niveau de votre GSI.
- Si Auto Scaling est activé (par défaut pour les tables créées dans la console) :
  - Optimisez les paramètres Auto Scaling d'écriture de votre GSI.

Techniques courantes de diagnostic et de surveillance

Lors de la résolution des erreurs de débit, plusieurs CloudWatch mesures peuvent aider à en identifier la cause première.

CloudWatch Indicateurs essentiels

Surveillez ces métriques clés pour diagnostiquer la limitation de la capacité provisionnée :

- Événements de limitation : `ReadProvisionedThroughputThrottleEvents` et `WriteProvisionedThroughputThrottleEvents` suivent les demandes qui sont limitées pour cette raison. `ReadThrottleEvents` et `WriteThrottleEvents` suivent les demandes de lecture ou d'écriture qui dépassent la capacité provisionnée.
- Consommation de capacité : `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits` affichent l'utilisation réelle.
- Capacité provisionnée : `ProvisionedReadCapacityUnits` et `ProvisionedWriteCapacityUnits` affichent les limites configurées.

Procédures de résolution

Augmentation de la capacité de débit des tables

Utilisez cette procédure lorsque Auto Scaling n'est pas activé et que vous avez besoin d'une augmentation de capacité immédiate.

1. Mettez à jour la capacité provisionnée de votre table à l'aide de la console AWS CLI DynamoDB ou du SDK :

   - Pour la capacité de lecture : augmentez le paramètre `ReadCapacityUnits`, qui indique le nombre maximum de lectures fortement cohérentes consommées par seconde avant que DynamoDB ne limite les demandes.
   - Pour la capacité d'écriture : augmentez le paramètre `WriteCapacityUnits`, qui indique le nombre maximum d'écritures consommées par seconde avant que DynamoDB ne limite les demandes.

2. Vérifiez que vos nouveaux paramètres de capacité ne dépassent pas les quotas de débit par table et que la consommation totale de votre compte reste inférieure aux quotas de débit par compte pour votre région. Si vous vous rapprochez de ces limites, envisagez plutôt de passer en mode de capacité à la demande.

Configuration de l'autoscaling des tables pour ajuster la capacité de lecture ou d'écriture de votre table ou de votre GSI

Configurez DynamoDB Auto Scaling pour ajuster automatiquement la capacité de lecture ou d'écriture en fonction des modèles de trafic. Vous pouvez configurer Auto Scaling indépendamment pour les deux tables GSIs, avec des contrôles distincts pour les unités de capacité de lecture et d'écriture.

1. Activez Auto Scaling pour la capacité de lecture, la capacité d'écriture ou les deux au niveau de votre table ou de votre GSI.
2. Définissez un pourcentage d'utilisation cible avec une marge de manœuvre pour les pics de trafic.

   > **Note**
   >
   > La baisse du taux d'utilisation cible augmente les coûts et la fréquence de mise à l'échelle. Les objectifs inférieurs à 40 % peuvent entraîner un surprovisionnement. Surveillez les modèles d'utilisation et les coûts pour trouver un juste milieu entre performance et efficacité.

3. Définissez les limites de capacité :

   - Minimum RCUs/WCUs: Maintient une capacité suffisante pendant les périodes de faible trafic.

- Maximum RCUs/WCUs: répond aux pics de trafic et protège contre les événements de montée en flèche.

Pour obtenir des conseils sur la configuration et la gestion de DynamoDB Auto Scaling, consultez Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

> **Note**
>
> Auto Scaling nécessite généralement plusieurs minutes avant de répondre aux modifications du trafic. En cas de pics de trafic soudains, la capacité de débordement de votre table fournit une protection immédiate pendant qu'Auto Scaling s'ajuste. Configurez l'utilisation cible avec une marge de manœuvre adéquate afin de laisser le temps de mettre à l'échelle les opérations et de préserver la capacité de débordement en cas de demande imprévue.

Optimisation des paramètres Auto Scaling de lecture ou d'écriture de votre table ou de votre index

Utilisez cette procédure lorsque Auto Scaling est activé, mais que la limitation persiste. Vous pouvez régler Auto Scaling indépendamment pour les tables et les index secondaires globaux (GSIs), avec des contrôles distincts pour les unités de capacité de lecture et d'écriture.

- Ajustez l'utilisation cible : envisagez de réduire l'utilisation cible de votre table ou GSIs de déclencher le dimensionnement plus tôt avant que la limitation ne se produise. Assurez-vous de surveiller le trafic après avoir effectué ces ajustements. Consultez the section called "Configuration de l'autoscaling des tables pour ajuster la capacité de lecture ou d'écriture de votre table ou de votre GSI" pour plus d'informations sur la consommation de capacité et les implications en matière de coûts.
- Vérifiez les limites de capacité : assurez-vous que vos paramètres de capacité minimale et maximale correspondent à vos modèles de charge de travail réels.

Passer au mode de capacité à la demande

Pour des informations générales sur la façon de changer de mode de capacité, consultez the section called "Changement de mode de capacité". Reportez-vous aux Service Quotas pour en savoir plus sur les contraintes spécifiques liées au changement de mode.

Augmentation de la capacité de débit du GSI

Utilisez cette procédure lorsque Auto Scaling n'est pas activé sur votre GSI ou si vous avez besoin d'une augmentation de capacité immédiate.

1. Mettez à jour la capacité provisionnée du GSI à l'aide de la console DynamoDB ou du SDK : AWS CLI

   - Pour la capacité de lecture : augmentez le paramètre `ReadCapacityUnits` de ce GSI spécifique, qui indique le nombre maximal de lectures que le GSI peut effectuer par seconde avant que DynamoDB ne limite les demandes. Notez que GSIs seules les lectures éventuellement cohérentes sont prises en charge.
   - Pour la capacité d'écriture : augmentez le paramètre `WriteCapacityUnits` pour ce GSI spécifique, qui indique le nombre maximum d'écritures que le GSI peut consommer par seconde avant que DynamoDB ne limite les demandes.
2. Assurez-vous que la capacité de débit provisionné du GSI reste dans les limites des quotas de débit par compte et par table.

Ressources supplémentaires

- Pour obtenir des informations détaillées sur la gestion des pics de trafic dans les tables de capacité provisionnée de DynamoDB, y compris les différentes stratégies allant de l'utilisation d'Auto Scaling et de la capacité de débordement à la gestion stratégique de l'accélérateur, consultez Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- Pour plus d'informations sur l'utilisation d'une expression cron pour planifier une politique de mise à l'échelle, consultez Optimize costs by scheduling provisioned capacity for DynamoDB.
- Pour obtenir des informations pratiques sur la surveillance et l'analyse des modèles d'utilisation du débit pour vos tables DynamoDB en mode de capacité provisionnée, consultez How to evaluate throughput utilization for Amazon DynamoDB tables in provisioned mode.

3- Dépassement des limites de compte

Les tables à la demande ne disposent pas de niveaux de capacité provisionnée à gérer, mais DynamoDB applique des limites de débit au niveau du compte afin d'empêcher une exécution incontrôlée et d'assurer une utilisation équitable des ressources pour tous les clients. Ces limites de compte par table jouent le rôle de garanties ajustables et sont définies pour chaque combinaison de compte et de région. Lorsque votre taux de consommation en lecture ou en écriture dépasse ces

limites, DynamoDB renvoie le type de raison de limitation `AccountLimitExceeded` dans l'exception de limitation. Les limites de compte par défaut par table s'appliquent automatiquement lorsqu'aucun paramètre de débit maximal personnalisé n'a été configuré pour les tables. Vous pouvez choisir de configurer des paramètres de débit maximal pour un meilleur contrôle des coûts et une meilleure prévisibilité, ou de demander des augmentations de quotas via la console the section called "Quotas" si les exigences de votre application dépassent les limites par défaut.

La limite du compte a dépassé les mesures d'atténuation

Cette section fournit des conseils de résolution pour les scénarios de limitation par rapport aux limites du compte. Avant d'utiliser ce guide, assurez-vous d'avoir identifié les raisons spécifiques de la limitation liées à la gestion des exceptions par votre application et d'avoir déterminé l'Amazon Resource Name (ARN) de la ressource concernée. Pour en savoir plus sur la façon de déterminer les raisons de la limitation et d'identifier des ressources limitées, consultez the section called "Cadre de diagnostic".

Avant de vous lancer dans des scénarios de limitation spécifiques, déterminez si une action est réellement nécessaire :

- Évaluez l'impact sur les performances : vérifiez si votre application répond toujours à ses exigences de performances malgré la limitation. De nombreuses applications fonctionnent sans problème lorsqu'elles s'approchent des limites du compte ou qu'elles les atteignent, notamment lors des opérations groupées ou des migrations de données.
- Passez en revue les modèles récurrents de limitation : si la limitation est temporaire et que votre application gère les nouvelles tentatives de manière efficace, les limites actuelles peuvent suffire à votre charge de travail.

Si votre application fonctionne de manière acceptable même lorsque les limites du compte sont parfois atteintes, vous pouvez choisir de simplement suivre de plus près la situation plutôt que de mettre en œuvre des modifications immédiates.

Si vous déterminez que la limitation est à l'origine de problèmes de performances ou de fiabilité inacceptables, sélectionnez une raison de limitation spécifique ci-dessous pour déterminer les options d'atténuation recommandées :

- the section called "TableReadAccountLimitExceeded"
- the section called "TableWriteAccountLimitExceeded"
- the section called "IndexReadAccountLimitExceeded"

- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

Quand cela se produit

La consommation en lecture de votre table a dépassé le quota de débit de lecture par table au niveau du compte pour votre région. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Procédez comme suit pour résoudre ce problème de limitation :

- Demandez une augmentation des quotas :

  Demandez une augmentation de la limite de débit de lecture par table (code de quota CBE56 L-CF0). Pour connaître les étapes détaillées de soumission de cette demande, consultez Demande d’augmentation du quota par table.

TableWriteAccountLimitExceeded

Quand cela se produit

La consommation en écriture de votre table a dépassé le quota de débit d’écriture par table au niveau du compte pour votre région. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Procédez comme suit pour résoudre ce problème de limitation :

- Demande d'augmentation du quota : demandez une augmentation de la limite de débit d'écriture par table (code de quota L-AB614373). Pour connaître les étapes détaillées de soumission de cette demande, consultez Demande d’augmentation du quota par table.

IndexReadAccountLimitExceeded

Quand cela se produit

Les opérations de lecture dirigées vers un index secondaire global (GSI) consomment plus de débit que ce que le quota de lecture par table de votre compte ne le permet dans votre région AWS actuelle. Le quota de débit de lecture par table au niveau du compte s'applique collectivement à une table et à toutes ses composantes combinées. GSIs Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Étapes à suivre pour résoudre le problème

Choisissez la solution appropriée en fonction de la répartition de la capacité de votre compte :

- Demande d'augmentation du quota : demandez une augmentation de la limite de débit de lecture par table (code de quota CBE56 L-CF0). Pour connaître les étapes détaillées de soumission de cette demande, consultez Demande d’augmentation du quota par table.
- Optimisez l’utilisation des GSI : passez en revue la conception des GSI et les modèles de requêtes afin de réduire toute consommation superflue de la capacité de lecture.

IndexWriteAccountLimitExceeded

Quand cela se produit

Les opérations d'écriture dans votre table de base génèrent des mises à jour correspondantes d'un GSI qui dépassent collectivement le quota de débit d'écriture par table au niveau du compte pour votre région. AWS Chaque écriture dans un élément de table de base contenant des attributs indexés par un GSI déclenche une opération d’écriture correspondante dans ce GSI. Ces opérations d’écriture combinées sont prises en compte dans votre quota de débit d’écriture par table. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” afin d'analyser les modèles et le calendrier de ces événements de régulation et d'identifier les opérations à l'origine de l'activité d'écriture GSI excessive.

Étapes à suivre pour résoudre le problème

Choisissez la solution appropriée en fonction de la répartition de la capacité de votre compte :

- Demande d'augmentation du quota : demandez une augmentation de la limite de débit d'écriture par table (code de quota L-AB614373) afin de répondre à l'augmentation du trafic d'écriture GSI provenant des opérations de table de base. Le quota de débit d'écriture par table s'applique à l'ensemble de la table, y compris à l'ensemble de celle-ci. GSIs Pour connaître les étapes

détaillées de soumission de cette demande, consultez Demande d'augmentation du quota par table.

- Optimisation des projections GSI : passez en revue les projections et la conception GSI afin de réduire le volume d'écriture à. GSIs

Techniques courantes de diagnostic et de surveillance

Lorsque vous résolvez les problèmes liés au dépassement de la limite du compte, plusieurs CloudWatch indicateurs peuvent vous aider à déterminer si vous atteignez des limites par table ou à l'échelle du compte et à comprendre vos modèles de distribution de capacité.

CloudWatch Indicateurs essentiels

Surveillez ces métriques clés pour diagnostiquer la limitation par rapport aux limites du compte :

- Événements de limitation par rapport aux limites du compte : `ReadAccountLimitThrottleEvents` et `WriteAccountLimitThrottleEvents` suivent les demandes qui sont limitées en raison des limites définies au niveau du compte. `ReadThrottleEvents` et `WriteThrottleEvents` suivent les demandes de lecture ou d'écriture qui dépassent la capacité provisionnée.
- Consommation de capacité par ressource : `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits` pour chaque table et GSI indique l'utilisation individuelle des ressources.
- Consommation à l'échelle du compte : utilisez des expressions mathématiques CloudWatch métriques pour additionner la capacité consommée dans toutes les tables et GSIs pour surveiller l'utilisation totale du compte.

Procédures de résolution

Demande d'augmentation du quota par table

Si vos applications doivent fonctionner au-delà des limites de débit par table actuelles, vous devez soumettre une demande d'augmentation de quota en suivant la procédure ci-dessous. Chaque table DynamoDB de AWS votre compte (ainsi que toutes les tables GSIs associées) est soumise à ces quotas de débit dans une région spécifique. Ces quotas représentent la capacité maximale de lecture ou d'écriture que chaque table individuelle et sa batterie GSIs peuvent consommer collectivement, et ils s'appliquent indépendamment à chaque table plutôt que sous forme d'agrégat pour toutes les tables de votre compte.

Vous pouvez choisir de définir des limites inférieures par table ou par GSI en configurant leurs paramètres de débit maximal à la demande.

1. Identifiez le quota spécifique qui doit être augmenté :
   - Limite de débit de lecture par table (code de quota L-CF0CBE56) : 40 000 par défaut par table RCUs
   - Limite de débit d'écriture par table (code de quota L-AB614373) : 40 000 WCUs par défaut par table
2. Utilisez la console AWS Service Quotas pour demander une augmentation de quota :
   - Accédez au service DynamoDB dans Service Quotas
   - Recherchez le quota approprié à l'aide du code correspondant
   - Demandez une augmentation en fonction de votre pic d'utilisation prévu
3. Justifiez l'augmentation, en fournissant par exemple les informations suivantes :
   - Schémas d'utilisation actuels et exigences relatives aux pics de trafic
   - Argumentaire justifiant l'augmentation de la capacité
   - Chronologie indiquant quand l'augmentation de la capacité est nécessaire

> **Note**
>
> Le traitement des augmentations de quotas dure généralement entre 24 et 48 heures. Planifiez vos demandes en conséquence et envisagez des stratégies d'atténuation temporaires en attendant leur approbation.

Optimisation des projections et de la conception des GSI

Optimisez les projections et la conception des index secondaires globaux (GSI) afin de réduire la consommation de capacité et d'améliorer les performances.

Stratégies de projection sélective

Si vos requêtes ne doivent accéder qu'à quelques attributs, la projection de ces seuls attributs réduit la quantité de données écrites dans le GSI lorsque les éléments de la table de base changent. Pour plus de détails sur les types de projection, consultez Projections for Global Secondary Indexes.

1. Analysez les modèles de requêtes : passez en revue les modèles de requête de votre application pour identifier les attributs réellement utilisés via le GSI.
2. Utilisez des projections sélectives : concentrez-vous uniquement sur les attributs de projet qui sont réellement nécessaires dans les requêtes pour réduire le volume d'écriture.
3. Envisagez de n'utiliser que les clés (`KEYS_ONLY`) : si vos requêtes ne nécessitent que les attributs de clés, utilisez la projection `KEYS_ONLY` pour minimiser le volume d'écriture.
4. Trouvez un juste milieu entre lecture et écriture : la projection d'un nombre réduit d'attributs réduit la consommation de la capacité d'écriture, mais peut nécessiter des lectures supplémentaires de la table de base.

Implémentation de GSI fragmentés

Sparse GSIs contient uniquement les éléments dotés de l'attribut indexé, plutôt que tous les éléments de votre table de base. Cela réduit la densité des partitions et améliore les performances lorsque vous interrogez fréquemment des sous-jeux de données spécifiques.

1. Conception GSIs qui inclut uniquement des éléments dotés de valeurs d'attributs spécifiques.
2. Implémentez l'indexation conditionnelle en ne définissant l'attribut de clé de partition des GSI que sur les éléments qui doivent être indexés.
3. Utilisez des clés composites en mode sparse GSIs (par exemple, status #timestamp) pour mieux répartir le trafic au sein du sous-ensemble d'éléments indexés.

Pour plus d'informations sur la mise en œuvre de ces stratégies, consultez Bonnes pratiques d'utilisation d'index secondaires dans DynamoDB.

4- Dépassement du débit maximal à la demande

Lorsque vous configurez une table ou un index secondaire global (GSI) à la demande, vous avez la possibilité de définir une limite de débit maximale (`MaxReadRequestUnits` et `MaxWriteRequestUnits`) au niveau de la table ou de l'index afin d'éviter que les coûts ne deviennent excessifs ou que les systèmes en aval ne soient submergés. Pour plus d'informations sur le débit maximal, consultez the section called "Débit maximal DynamoDB pour les tables à la demande".

Lorsque votre consommation en lecture ou en écriture dépasse ces limites que vous vous imposez vous-même, les demandes supplémentaires qui dépassent ces limites reçoivent des réponses

de limitation rapides. DynamoDB renvoie des exceptions avec le type de raison de limitation `MaxOnDemandThroughputExceeded`, indiquant quelle ressource a atteint sa limite de débit.

Le débit maximal à la demande a dépassé la limitation

Cette section fournit des conseils pour résoudre les problèmes de limitation lorsque le débit maximal à la demande est dépassé. Avant d'utiliser ce guide, assurez-vous d'avoir identifié les raisons spécifiques de la limitation liées à la gestion des exceptions par votre application et d'avoir déterminé l'Amazon Resource Name (ARN) de la ressource concernée. Pour en savoir plus sur la façon de déterminer les raisons de la limitation et d'identifier des ressources limitées, consultez the section called "Cadre de diagnostic".

Avant de vous lancer dans des scénarios de limitation spécifiques, déterminez si une action est réellement nécessaire :

- Évaluez vos paramètres de débit maximal : ces limites ont été configurées intentionnellement pour contrôler les coûts ou protéger les systèmes en aval. Si vous recevez des événements de limitation de type `MaxOnDemandThroughputExceeded`, ces limites fonctionnent comme prévu. Déterminez si l'augmentation de ces limites correspond à vos objectifs initiaux de contrôle des coûts ou de protection des systèmes.
- Évaluez l'impact sur les applications : déterminez si la limitation cause réellement des problèmes à vos applications ou à vos utilisateurs. Si vos applications gèrent efficacement les nouvelles tentatives et répondent à leurs exigences de performances malgré des limitations occasionnelles, le maintien de vos limites actuelles peut être le choix approprié.
- Passez en revue les modèles de trafic : déterminez si la limitation représente un modèle de trafic attendu ou un pic inhabituel. Pour les modèles de trafic prévisibles et récurrents qui dépassent constamment vos limites, il peut être justifié d'ajuster les paramètres de débit maximal. Pour les pics temporaires, il peut être plus approprié de mettre en œuvre de meilleures stratégies de distribution des demandes que d'augmenter les limites.

Si, après mûre réflexion, vous déterminez que vos paramètres de débit maximal doivent être ajustés, reportez-vous aux scénarios de limitation spécifiques ci-dessous pour déterminer les options de correction ciblées :

- the section called "TableReadMaxOnDemandThroughputExceeded"
- the section called "TableWriteMaxOnDemandThroughputExceeded"
- the section called "IndexReadMaxOnDemandThroughputExceeded"

- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

Quand cela se produit

Votre table à la demande a dépassé sa capacité de débit de lecture maximale configurée. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Augmentez la limite de débit maximal : utilisez la console DynamoDB, l’AWS CLI ou l’API DynamoDB `UpdateTable` pour augmenter la valeur `MaxReadRequestUnits` pour la table concernée, puis surveillez ce qui se passe et procédez aux ajustements nécessaires. Cela permet à la table de gérer un débit de lecture plus élevé avant que la limitation ne se produise.
- Supprimez la limite maximale : définissez `MaxReadRequestUnits` sur `-1` pour supprimer le plafond, afin de permettre une mise à l’échelle en fonction de la demande dans les limites de quotas de débit de votre compte. Cela supprime votre limite personnalisée tout en respectant les quotas définis au niveau du compte AWS. Cependant, il est important de surveiller les dépenses de près après avoir supprimé cette limite, car votre table peut désormais consommer beaucoup plus de capacité avant d’atteindre les quotas fixés au niveau du compte.

TableWriteMaxOnDemandThroughputExceeded

Quand cela se produit

Votre table à la demande a dépassé sa capacité de débit d’écriture maximale configurée. Vous pouvez surveiller les CloudWatch métriques the section called “Techniques courantes de diagnostic et de surveillance” pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Augmentez la limite de débit maximal : utilisez la console DynamoDB, l’AWS CLI ou l’API DynamoDB `UpdateTable` pour augmenter la valeur `MaxWriteRequestUnits` pour la table concernée, puis surveillez ce qui se passe et procédez aux ajustements nécessaires.

- Supprimez la limite maximale : définissez `MaxWriteRequestUnits` sur `-1` pour supprimer le plafond, afin de permettre une mise à l'échelle en fonction de la demande dans les limites de quotas de débit de votre compte. Cela supprime votre limite personnalisée tout en respectant les quotas définis au niveau du compte AWS. Cependant, il est important de surveiller les dépenses de près après avoir supprimé cette limite, car votre table peut désormais consommer beaucoup plus de capacité avant d'atteindre les quotas fixés au niveau du compte.

IndexReadMaxOnDemandThroughputExceeded

Quand cela se produit

Les demandes de lecture adressées à un GSI en mode à la demande ont dépassé la capacité de débit de lecture maximale configurée pour les GSI. Vous pouvez surveiller les CloudWatch métriques the section called "Techniques courantes de diagnostic et de surveillance" pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Augmentez la limite de débit maximal pour les GSI : utilisez la console DynamoDB, l'AWS CLI ou l'API DynamoDB `UpdateTable` pour augmenter la valeur `MaxReadRequestUnits` pour le GSI concerné, puis surveillez ce qui se passe et procédez aux ajustements nécessaires.
- Supprimez la limite maximale pour les GSI : définissez `MaxReadRequestUnits` sur `-1` pour supprimer le plafond pour les GSI, afin de permettre une mise à l'échelle en fonction de la demande dans les limites de quotas de débit de votre compte. Cela supprime votre limite personnalisée tout en maintenant les garanties au niveau AWS du compte. Toutefois, une fois que vous avez supprimé cette limite, il est important de suivre les dépenses de près.

IndexWriteMaxOnDemandThroughputExceeded

Quand cela se produit

Les mises à jour des éléments de la table de base déclenchent des écritures dans un GSI en mode à la demande. Comme elles dépassent la capacité de débit d'écriture maximale configurée pour les GSI, cela entraîne une limitation de contre-pression. Vous pouvez surveiller les CloudWatch métriques the section called "Techniques courantes de diagnostic et de surveillance" pour analyser votre événement de régulation.

Options de correction

Envisagez les étapes suivantes pour gérer les problèmes de limitation :

- Augmentez la limite de débit maximal pour les GSI : utilisez la console DynamoDB, l'AWS CLI ou l'API DynamoDB `UpdateTable` pour augmenter la valeur `MaxWriteRequestUnits` pour le GSI concerné, puis surveillez ce qui se passe et procédez aux ajustements nécessaires.
- Supprimez la limite maximale pour les GSI : définissez `MaxWriteRequestUnits` sur `-1` pour supprimer le plafond pour les GSI, afin de permettre une mise à l'échelle en fonction de la demande dans les limites de quotas de débit de votre compte. Cela supprime votre limite personnalisée tout en maintenant les garanties au niveau AWS du compte. Toutefois, une fois que vous avez supprimé cette limite, il est important de suivre les dépenses de près.

Techniques courantes de diagnostic et de surveillance

Lorsque le débit maximal à la demande est dépassé, plusieurs CloudWatch indicateurs peuvent aider à identifier la cause première et les modèles de dimensionnement.

CloudWatch Indicateurs essentiels

Surveillez ces métriques clés pour diagnostiquer le dépassement de la limitation par rapport au débit maximal à la demande :

- Événements de limitation du débit maximal : `ReadMaxOnDemandThroughputThrottleEvents` et `WriteMaxOnDemandThroughputThrottleEvents` suivent les demandes qui sont limitées en raison du dépassement des limites maximales. `ReadThrottleEvents` et `WriteThrottleEvents` suivent les demandes de lecture ou d'écriture qui dépassent la capacité allouée.
- Débit maximal actuel configuré pour une table ou un index secondaire global : `OnDemandMaxReadRequestUnits` et `OnDemandMaxWriteRequestUnits` affichent les limites de capacité maximale actuelles.
- Consommation de capacité réelle : `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits` affichent les modèles d'utilisation réels.

Approche d'analyse

Suivez ces étapes pour confirmer le diagnostic de dépassement du débit maximal à la demande :

1. Comparez la capacité consommée aux limites de capacité maximale : vérifiez si la consommation se rapproche des limites maximales ou les dépasse régulièrement.
2. Passez en revue la fréquence et le calendrier des événements de limitation pour identifier des modèles récurrents. Identifiez toute augmentation soudaine de la capacité consommée qui coïnciderait avec votre événement de limitation.
3. Utilisez CloudWatch Contributor Insights pour identifier les éléments ou les clés de partition qui consomment le plus de capacité.

Compréhension de la limitation d'écriture et de la contre-pression des index secondaires globaux (GSI) dans DynamoDB

La limitation de contre-pression des GSI représente l'un des scénarios de limitation les plus complexes de DynamoDB, car elle crée une relation indirecte entre les opérations d'écriture et la limitation : votre application écrit dans une table de base mais subit une limitation en raison de contraintes de capacité sur un ou plusieurs index.

Compréhension de la limitation de contre-pression des GSI

Lorsque vous écrivez dans une table DynamoDB, tous les index secondaires globaux GSIs () de cette table sont mis à jour de manière asynchrone à l'aide d'un modèle finalement cohérent. Si un des GSI ne dispose pas d'une capacité suffisante pour gérer ces mises à jour, DynamoDB limite les écritures dans la table de base afin de garantir la cohérence des données. C'est ce qu'on appelle la contre-pression de GSI. Pour plus d'informations sur le GSIs fonctionnement, voir Index secondaires globaux dans DynamoDB.

Contrairement à la limitation directe des tables où la ressource à laquelle on accède est également la ressource à l'origine de la limitation, la contre-pression des GSI crée une dépendance entre la table de base et ses index. Même si la table de base dispose d'une grande capacité, les écritures sont limitées si un des GSI associés ne peut pas gérer le volume de mise à jour. Il est particulièrement important de comprendre cette relation, car les contraintes au niveau de la partition s'appliquent indépendamment à la fois à la table de base et à chacun des GSI, chacun ayant sa propre structure de partition et des limites de débit correspondantes.

Le partitionnement de GSI est basé sur la clé de partition des GSI, qui est souvent différente de la clé de partition de la table de base. Même si l'accès à la table de base est parfaitement réparti entre les partitions, les mises à jour de GSI peuvent tout de même se concentrer sur des partitions spécifiques, créant ainsi des points chauds dans les GSI. Pour connaître les meilleures pratiques générales

relatives à la conception des clés de partition pour les deux tables GSIs, voir Conception des clés de partition DynamoDB.

Par exemple, si la table de base utilise `customerId` comme clé de partition (répartie uniformément) mais que les GSI utilisent `status` comme clé de partition (avec des valeurs possibles limitées telles que « actif », « en attente », « fermé »), les mises à jour d'éléments dont les valeurs de statut sont populaires peuvent créer des partitions critiques de GSI, même lorsque l'accès à la table de base est équilibré. Cela crée un scénario particulièrement difficile dans lequel votre application peut subir une limitation en raison de partitions critiques de GSI, même si la table de base et les GSI disposent d'une capacité globale suffisante et que le modèle d'accès à la table de base semble bien distribué.

Même si l'exception de limitation pointe vers les GSI (via `ResourceArn`), l'opération réelle qui fait l'objet d'une limitation est l'écriture dans la table de base. Cela peut être source de confusion, car votre application écrit dans la table de base mais reçoit une exception concernant les GSI.

Types de limitation de GSI

La limitation de contre-pression des GSI se manifeste par différents types d'exception en fonction de la contrainte de capacité spécifique :

- Dépassement de la capacité provisionnée de GSI : survient lorsque les GSI ne disposent pas d'unités de capacité d'écriture suffisantes pour gérer les mises à jour à partir des opérations de la table de base. Cela produit une `ProvisionedThroughputExceededException` avec la raison `IndexWriteProvisionedThroughputExceeded`, et le `ResourceArn` indique directement que les GSI spécifiques sont confrontés à des contraintes de capacité.
- Dépassement du débit maximal à la demande de GSI : survient lorsque les opérations d'écriture de GSI dépassent les limites maximales configurées sur les tables à la demande. Cela produit une `ThrottlingException` avec la raison `IndexWriteMaxOnDemandThroughputExceeded`, qui identifie les GSI spécifiques avec des restrictions de débit configurées.
- Dépassement des limites de partition de GSI : se produit lorsque des partitions de GSI individuelles dépassent leurs limites de débit (partitions critiques), même si la capacité de GSI globale semble suffisante. Cela génère une `ThrottlingException` avec la raison `IndexWriteKeyRangeThroughputExceeded`, ce qui indique des problèmes de partition chaudes sur le GSI spécifique identifié dans le `ResourceArn`. Cela est particulièrement important, car la distribution des partitions de GSI peut différer considérablement de la distribution des partitions de la table de base, ce qui crée des points chauds dans les GSI même lorsque l'accès à la table de base est réparti de manière uniforme.

- Dépassement des limites de compte de GSI : se déclenche lorsque les opérations d'écriture dans un GSI spécifique dépassent les limites de débit régionales par table (ou un GSI individuel dans cette table) définies au niveau du compte. DynamoDB renvoie une `ThrottlingException` avec la raison `IndexWriteAccountLimitExceeded`, pointant vers le GSI qui a poussé son utilisation au-delà des limites du compte. Cette limitation se produit indépendamment pour chaque GSI qui dépasse la limite. Pour en savoir plus sur les quotas de service par table, par compte et par région, consultez the section called "Quotas".

Métriques de limitation CloudWatch

Cette page fournit un guide complet des métriques CloudWatch est spécifiquement conçues pour vous aider à identifier, diagnostiquer et résoudre les problèmes de limitation dans les tables et index DynamoDB.

Métriques de limitation générales

- `ThrottledRequests`
  - Incrémenté d'une unité lors de la limitation d'un événement d'une demande, quel que soit le nombre d'événements individuels limités pour cette demande. Par exemple, la mise à jour d'un élément dans une table ayant des GSI génère plusieurs événements : une opération d'écriture dans la table de base et une opération d'écriture dans chaque index. Si l'un de ces événements individuels est limité, la métrique `ThrottledRequests` n'est incrémentée qu'une seule fois.

    Il est important de comprendre ce comportement lors de la surveillance et du dépannage des performances de DynamoDB, car il peut masquer l'étendue réelle de la limitation. Pour obtenir des informations plus complètes, comparez la métrique `ThrottledRequests` avec les métriques spécifiques au niveau de l'événement, tels que `ReadThrottleEvents`, `WriteThrottleEvents` et des métriques ciblées comme `ReadKeyRangeThroughputThrottleEvents`. La liste complète de ces métriques propres à une cause est disponible sur cette page. Chaque métrique correspond à des raisons de limitation particulières qui sont capturées dans l'exception correspondante. Pour en savoir plus sur la façon de récupérer et d'interpréter ces raisons lors d'événements de limitation, consultez la section the section called "Diagnostic des problèmes de limitation" qui fournit des instructions pour identifier et résoudre les causes racines des problèmes de limitation.
- `ReadThrottleEvents`

    - Surveillez les demandes dépassant le RCU alloué pour une table ou un index secondaire global.
- `WriteThrottleEvents`
    - Surveillez les demandes qui dépassent le nombre de WCU allouées pour une table ou un index secondaire global.

Mesures de limitation détaillées par cause

Limitation du débit à la demande

- `ReadMaxOnDemandThroughputThrottleEvents`
    - Nombre de demandes de lecture limitées en raison du débit maximal à la demande.
- `WriteMaxOnDemandThroughputThrottleEvents`
    - Nombre de demandes d'écriture limitées en raison du débit maximal à la demande.

Limitation au niveau du compte

- `ReadAccountLimitThrottleEvents`
    - Nombre de demandes de lecture limitées en raison des limites du compte.
- `WriteAccountLimitThrottleEvents`
    - Nombre de demandes d'écriture limitées en raison des limites du compte.

Limitation au niveau de la partition

- `ReadKeyRangeThroughputThrottleEvents`
    - Nombre de demandes de lecture limitées en raison des limites de partition.
- `WriteKeyRangeThroughputThrottleEvents`
    - Nombre de demandes d'écriture limitées en raison des limites de partition.

Métriques d'analyse de la capacité

- `OnlineIndexConsumedWriteCapacity`
    - Lorsque vous ajoutez un nouveau GSI à une table existante, DynamoDB effectue une opération de remplissage qui copie les données de la table de base dans le nouvel index. Ce processus consomme des unités de capacité d'écriture. La métrique `OnlineIndexConsumedWriteCapacity` suit cette consommation spécifique.

Cette consommation est distincte et s'ajoute aux opérations d'écriture régulières suivies par `ConsumedWriteCapacityUnits`. La métrique `ConsumedWriteCapacityUnits` régulière correspondant au GSI n'inclut pas le débit d'écriture consommé pendant de la création de l'index.

- `ProvisionedReadCapacityUnits` et `ProvisionedWriteCapacityUnits`
  - Affichez le nombre d'unités de capacité de lecture ou d'écriture allouées qui ont été consommées durant la période spécifiée, pour une table ou un index secondaire global spécifié.
  - Notez que la dimension `TableName` renvoie `ProvisionedReadCapacityUnits` pour la table uniquement par défaut. Pour afficher le nombre d'unités de capacité de lecture ou d'écriture allouées pour un index secondaire global, vous devez spécifier `TableName` et `GlobalSecondaryIndexName`.
- `ConsumedReadCapacityUnits` et `ConsumedWriteCapacityUnits`
  - Consultez le nombre d'unités de capacité de lecture ou d'écriture qui ont été consommées durant la période spécifiée. `ConsumedWriteCapacityUnits` n'inclut pas la capacité d'écriture consommée lors du processus initial de création de l'index.

Pour plus d'informations sur les métriques DynamoDB CloudWatch, consultez Métriques et dimensions DynamoDB.

Annexe DynamoDB

Rubriques

- Résolution des problèmes d'établissement de SSL/TLS connexion avec DynamoDB
- Exemples de tables et de données à utiliser dans DynamoDB
- Création d'exemples de tables et chargement de données dans DynamoDB
- Exemple d'application DynamoDB utilisant :AWS SDK pour Python (Boto) Tic-tac-toe
- Mots réservés dans DynamoDB
- AWS Exemples de SDK pour Java 1.x
- AWS Exemples de SDK pour Go 1.x
- AWS Exemples de SDK pour Node.js 2.x

Résolution des problèmes d'établissement de SSL/TLS connexion avec DynamoDB

Amazon DynamoDB est en train de déplacer nos points de terminaison vers des certificats sécurisés signés par l'autorité de certification Amazon Trust Services (ATS) au lieu d'une autorité de certification tierce. En décembre 2017, nous avons lancé la région EU-WEST-3 (Paris) avec les certificats sécurisés émis par Amazon Trust Services. Toutes les nouvelles régions lancées après décembre 2017 ont des points de terminaison avec les certificats émis par Amazon Trust Services. Ce guide explique comment valider une connexion SSL/TLS et résoudre des problèmes de connexion SSL/TLS.

Test de votre application ou service

La plupart AWS SDKs des interfaces de ligne de commande (CLIs) prennent en charge l'autorité de certification Amazon Trust Services. Si vous utilisez une version du AWS SDK pour Python ou de la CLI publiée avant le 29 octobre 2013, vous devez effectuer une mise à niveau. .NET, Java, PHP JavaScript, Go SDKs et C++ CLIs ne regroupent aucun certificat, leurs certificats proviennent du système d'exploitation sous-jacent. Le SDK Ruby inclut au moins l'un des éléments requis CAs depuis le 10 juin 2015. Avant cette date, le kit SDK Ruby V2 ne n'incluait pas de certificat. Si vous utilisez une version non prise en charge, personnalisée ou modifiée du AWS SDK, ou si vous utilisez un trust store personnalisé, il se peut que vous ne disposiez pas du support nécessaire pour Amazon Trust Services Certificate Authority.

Pour valider l'accès aux points de terminaison DynamoDB, vous devez développer un test qui accède à l'API DynamoDB ou à l'API DynamoDB Streams dans la région EU-WEST-3, et valider la réussite de la liaison TLS. Les points de terminaison spécifiques auxquels vous allez devoir accéder dans le cadre d'un tel test sont les suivants :

- DynamoDB : https://dynamodb.eu-west-3.amazonaws.com
- Streams DynamoDB : https://streams.dynamodb.eu-west-3.amazonaws.com

Si votre application ne prend pas en charge l'autorité de certification Amazon Trust Services, vous constaterez l'un des échecs suivants :

- SSL/TLS Erreurs de négociation
- Il s'agit d'un long délai avant que votre logiciel ne reçoive une erreur indiquant un échec de SSL/TLS négociation. Le délai dépend de la politique de nouvelle tentative et de la configuration du délai d'expiration de votre client.

Test de votre navigateur client

Pour vérifier que votre navigateur peut se connecter à Amazon DynamoDB, ouvrez l'URL suivante : https://dynamodb.eu-west-3.amazonaws.com Si le test réussit, vous voyez un message comme celui-ci :

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Si le test échoue, un message d'erreur similaire à celui-ci s'affichera https://untrusted-root.badssl.com:/.

Mise à jour de votre client d'application logicielle

Les applications accédant aux points de terminaison de l'API DynamoDB ou DynamoDB Streams (que ce soit par le biais de navigateurs ou par programmation) devront mettre à jour la liste des autorités de certification approuvées sur les machines clientes si elles ne prennent pas déjà en charge les éléments suivants : CAs

- Amazon Root CA 1
- Starfield Services Root Certificate Authority – G2
- Starfield Class 2 Certification Authority

Si les clients font déjà confiance à l'une des trois options ci-dessus CAs , ceux-ci feront confiance aux certificats utilisés par DynamoDB et aucune action n'est requise. Toutefois, si vos clients ne font déjà confiance à aucune des solutions ci-dessus CAs, les connexions HTTPS aux flux DynamoDB ou DynamoDB échoueront. APIs Pour plus d'informations, consultez ce billet de blog : https:// aws.amazon.com/blogs/security/ how-to-prepare-for - aws-move-to-its -own-certificate-authority/.

Mise à jour de votre navigateur client

Vous pouvez mettre à jour la solution groupée de certificats dans votre navigateur simplement en mettant à jour votre navigateur. Vous pouvez trouver des instructions pour les navigateurs les plus courants sur les sites web des navigateurs :

- Chrome : https://support.google.com/chrome/réponse/95414 ? hl=fr
- Firefox : https://support.mozilla. org/en-US/kb/update- firefox-latest-version
- Safari : https://support.apple.com/en-us/HT20416
- Internet Explorer : https://support.microsoft.com/en-us/aide/17295/ windows-internet-explorer-which -version #ie =autre

Mise à jour manuelle de votre solution groupée de certificats

Si vous ne pouvez pas accéder à l’API DynamoDB ou DynamoDB Streams, vous devez mettre à jour votre offre groupée de certificats. Pour ce faire, vous devez importer au moins l'un des éléments requis CAs. Vous pouvez les trouver sur https://www.amazontrust.com/repository/.

Les systèmes d’exploitation et les langages de programmation suivants prennent en charge les certificats Amazon Trust Services :

- Versions de Microsoft Windows sur lesquelles des mises à jour de janvier 2005 ou ultérieures sont installées, Windows Vista, Windows 7, Windows Server 2008 et versions ultérieures.
- macOS X 10.4 avec Java pour macOS X 10.4 version 5, macOS X 10.5 et versions ultérieures.
- Red Hat Enterprise Linux 5 (mars 2007), Linux 6 et Linux 7, et CentOS 5, CentOS 6 et CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (toutes versions)
- Java 1.4.2_12, Java 5 mise à jour 2 et toutes les versions plus récentes, dont Java 6, Java 7 et Java 8

Si vous ne parvenez toujours pas à vous connecter, consultez la documentation de votre logiciel, le fournisseur du système d'exploitation ou contactez le AWShttps://aws.amazon.comSupport/support pour obtenir de l'aide.

Exemples de tables et de données à utiliser dans DynamoDB

Le Guide du développeur Amazon DynamoDB utilise des exemples de tables pour illustrer divers aspects de DynamoDB.

Nom de la table	Clé primaire
ProductCatalog	Clé primaire simple : • `Id` (nombre).
Forum	Clé primaire simple : • `Name` (chaîne)
Thread	Clé primaire composite : • `ForumName` (chaîne) • `Subject` (chaîne)
Reply	Clé primaire composite : • `Id` (chaîne) • `ReplyDateTime` (chaîne)

La table Reply contient un index secondaire global nommé PostedBy-Message-Index. Cet index facilite les requêtes sur deux attributs autres que de clé de la table Reply.

Nom d'index	Clé primaire
PostedBy-Message-Index	Clé primaire composite : • `PostedBy` (chaîne) • `Message` (chaîne)

Pour plus d'informations sur ces tables, consultez Étape 1 : création d'une table dans DynamoDB et Étape 2 : écrire des données dans une table DynamoDB.

Fichiers de données d'exemple

Rubriques

- Exemple de données de ProductCatalog
- Exemple de données de Forum
- Exemple de données de Thread
- Exemple de données de Reply

Les sections suivantes présentent les exemples de fichiers de données utilisés pour charger les tables ProductCatalog, Forum, Thread et Replytables.

Chaque fichier de données contient plusieurs éléments `PutRequest` contenant chacun un seul élément. Ces éléments `PutRequest` sont utilisés en entrée pour l'opération `BatchWriteItem` à l'aide de l'AWS Command Line Interface (AWS CLI).

Exemple de données de ProductCatalog

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            }
                        ]
```

```
                },
                "Price": {
                    "N": "2"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.5"
                },
                "PageCount": {
                    "N": "500"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "102"
                },
                "Title": {
                    "S": "Book 102 Title"
                },
                "ISBN": {
                    "S": "222-2222222222"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
                            "S": "Author2"
                        }
                    ]
                },
                "Price": {
                    "N": "20"
                },
```

```
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.8"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "103"
                    },
                    "Title": {
                        "S": "Book 103 Title"
                    },
                    "ISBN": {
                        "S": "333-3333333333"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2000"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 1.5"
                    },
                    "PageCount": {
```

```
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": false
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "201"
                },
                "Title": {
                    "S": "18-Bike-201"
                },
                "Description": {
                    "S": "201 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Mountain A"
                },
                "Price": {
                    "N": "100"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
```

```
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
                },
                "Price": {
                    "N": "200"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
```

```
                "Id": {
                    "N": "203"
                },
                "Title": {
                    "S": "19-Bike-203"
                },
                "Description": {
                    "S": "203 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "300"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "204"
                },
                "Title": {
```

```
                    "S": "18-Bike-204"
                },
                "Description": {
                    "S": "204 Description"
                },
                "BicycleType": {
                    "S": "Mountain"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "400"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "205"
                },
                "Title": {
                    "S": "18-Bike-204"
                },
                "Description": {
                    "S": "205 Description"
                },
                "BicycleType": {
                    "S": "Hybrid"
                },
                "Brand": {
                    "S": "Brand-Company C"
```

```
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

Exemple de données de Forum

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
```

```
                }
            }
        }
    ]
}
```

Exemple de données de Thread

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "index"
                            },
                            {
```

```
                            "S": "primarykey"
                        },
                        {
                            "S": "table"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon DynamoDB"
                },
                "Subject": {
                    "S": "DynamoDB Thread 2"
                },
                "Message": {
                    "S": "DynamoDB thread 2 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-15T19:58:22.514Z"
                },
                "Views": {
                    "N": "3"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "items"
                        },
                        {
                            "S": "attributes"
```

```
                        },
                        {
                            "S": "throughput"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon S3"
                },
                "Subject": {
                    "S": "S3 Thread 1"
                },
                "Message": {
                    "S": "S3 thread 1 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-29T19:58:22.514Z"
                },
                "Views": {
                    "N": "0"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "largeobjects"
                        },
                        {
                            "S": "multipart upload"
                        }
```

```
                        ]
                    }
                }
            }
        }
    ]
}
```

Exemple de données de Reply

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-22T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 2 text"
                    },
                    "PostedBy": {
```

```
                        "S": "User B"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-29T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-10-05T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        }
    ]
}
```

Création d’exemples de tables et chargement de données dans DynamoDB

Cette annexe fournit du code pour créer les tables et ajouter des données par programme.

Rubriques

- Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour Java
- Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour .NET

Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour Java

L’exemple de code Java suivant crée des tables et charge des données dans celles-ci. Pour step-by-step obtenir des instructions sur l'exécution de ce code à l'aide d'Eclipse, consultezExemples de code Java.

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
```

```
public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);

        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
```

```
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
            table.waitForDelete();

        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

        try {

            ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
            keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                            // key

            ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
            attributeDefinitions
                    .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                            .withAttributeType(partitionKeyType));
```

```
            if (sortKeyName != null) {
                keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                                  // key
                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
            }

            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition

                                // key
                                new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                                  // key
                        .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }
```

```
            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102
Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x
0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103
 Title")
                    .withString("ISBN", "333-3333333333")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x
 1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
```

```
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
 Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Name", "Amazon
S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
"DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
A").withString("LastPostedDateTime", dateFormatter.format(date2))
```

```
                    .withNumber("Views", 0).withNumber("Replies",
0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"primarykey", "table")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
```

```
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
text").withString("PostedBy", "User B");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
```

```
                    .withString("Message", "DynamoDB Thread 2 Reply 2
 text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());

        }
    }

}
```

Création de tableaux d'exemple et téléchargement de données à l'aide du AWS SDK pour .NET

L'exemple de code C# suivant crée des tables et charge des données dans celles-ci. Pour step-by-step obtenir des instructions sur l'exécution de ce code dans Visual Studio, consultezExemples de code .NET.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
```

```
            DeleteTable("Thread");
            DeleteTable("Reply");

            // Create tables (using the AWS SDK for .NET low-level API).
            CreateTableProductCatalog();
            CreateTableForum();
            CreateTableThread(); // ForumTitle, Subject */
            CreateTableReply();

            // Load data (using the .NET SDK document API)
            LoadSampleProducts();
            LoadSampleForums();
            LoadSampleThreads();
            LoadSampleReplies();
            Console.WriteLine("Sample complete!");
            Console.WriteLine("Press ENTER to continue");
            Console.ReadLine();
        }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void DeleteTable(string tableName)
    {
        try
        {
            var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitTillTableDeleted(client, tableName, deleteTableResponse);
        }
        catch (ResourceNotFoundException)
        {
            // There is no such table.
        }
    }

    private static void CreateTableProductCatalog()
    {
        string tableName = "ProductCatalog";

        var response = client.CreateTable(new CreateTableRequest
        {
```

```
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Id",
                                AttributeType = "N"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "Id",
                                KeyType = "HASH"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableForum()
    {
        string tableName = "Forum";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Name",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
```

```
                                new KeySchemaElement
                                {
                                    AttributeName = "Name", // forum Title
                                    KeyType = "HASH"
                                }
                            },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableThread()
    {
        string tableName = "Thread";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "ForumName", // Hash attribute
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "Subject",
                                    AttributeType = "S"
                                }
                            },
            KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "ForumName", // Hash attribute
                                    KeyType = "HASH"
                                },
                                new KeySchemaElement
```

```
                                {
                                    AttributeName = "Subject", // Range attribute
                                    KeyType = "RANGE"
                                }
                            },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableReply()
    {
        string tableName = "Reply";
        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Id",
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "ReplyDateTime",
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "PostedBy",
                                    AttributeType = "S"
                                }
                            },
            KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement()
                                {
                                    AttributeName = "Id",
```

```
                                    KeyType = "HASH"
                                },
                                new KeySchemaElement()
                                {
                                    AttributeName = "ReplyDateTime",
                                    KeyType = "RANGE"
                                }
                            },
                LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
                            {
                                new LocalSecondaryIndex()
                                {
                                    IndexName = "PostedBy_index",


                                    KeySchema = new List<KeySchemaElement>() {
                                        new KeySchemaElement() {
                                            AttributeName = "Id", KeyType = "HASH"
                                        },
                                        new KeySchemaElement() {
                                            AttributeName = "PostedBy", KeyType =
"RANGE"

                                        }
                                    },
                                    Projection = new Projection() {
                                        ProjectionType = ProjectionType.KEYS_ONLY
                                    }
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
tableName,
                                 CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;
```

```
            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
tableName,
                             DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable
            try
            {
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
```

```
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books ********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);
```

```
            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);

            // ************ Add bikes. ******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
```

```
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);

            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;
```

```
            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0));
            thread1["Views"] = 0;
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };

            threadTable.PutItem(thread2);

            // Thread 3.
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
```

```
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }

        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");

            // Reply 1 - thread 1.
            var thread1Reply1 = new Document();
            thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
            thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
            thread1Reply1["PostedBy"] = "User A";

            replyTable.PutItem(thread1Reply1);

            // Reply 2 - thread 1.
            var thread1reply2 = new Document();
            thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
            thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
            thread1reply2["PostedBy"] = "User B";

            replyTable.PutItem(thread1reply2);

            // Reply 3 - thread 1.
            var thread1Reply3 = new Document();
            thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
            thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
            thread1Reply3["PostedBy"] = "User B";
```

```
            replyTable.PutItem(thread1Reply3);

            // Reply 1 - thread 2.
            var thread2Reply1 = new Document();
            thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
 0, 0, 0)); // Range attribute.
            thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
            thread2Reply1["PostedBy"] = "User A";


            replyTable.PutItem(thread2Reply1);

            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

Exemple d'application DynamoDB utilisant :AWS SDK pour Python (Boto) Tic-tac-toe

Le Tic-Tac-Toe jeu est un exemple d'application Web basée sur Amazon DynamoDB. L'application utilise le AWS SDK pour Python (Boto) pour effectuer les appels DynamoDB nécessaires pour stocker les données de jeu dans une table DynamoDB, et le framework Web Python Flask pour end-to-end illustrer le développement d'applications dans DynamoDB, y compris la façon de modéliser les données. Il démontre également de bonnes pratiques de modélisation des données dans DynamoDB, incluant la table que vous créez pour l'application de jeu, la clé primaire que vous définissez, les index supplémentaires dont vous avez besoin en fonction de vos exigences de requête, et l'utilisation d'attributs de valeurs concaténées.

Vous pouvez exécuter l' Tic-Tac-Toeapplication sur le Web de la manière suivante :

1. Vous vous connectez à la page d'accueil de l'application.
2. Ensuite, vous invitez un autre utilisateur à jouer au jeu en tant que votre adversaire.

   Jusqu'à ce qu'un autre utilisateur accepte votre invitation, le statut de jeu reste `PENDING`. Une fois qu'un adversaire accepte l'invitation, le statut du jeu passe à `IN_PROGRESS`.
3. Le jeu commence après que l'adversaire se connecte et accepte l'invitation.
4. L'application stocke toutes les informations de statuts et de mouvements du jeu dans une table DynamoDB.
5. Le jeu se termine par une victoire ou un nul, qui définit le statut du jeu sur `FINISHED`.

L'exercice de création de l' end-to-endapplication est décrit en plusieurs étapes :

- Étape 1 : déploiement et test localement – Dans cette section, vous téléchargez, déployez et testez l'application sur votre ordinateur local. Vous allez créer les tables requises dans la version téléchargeable de DynamoDB.
- Étape 2 : examen du modèle de données et des détails de la mise en œuvre – Cette section décrit tout d'abord en détails le modèle de données, y compris les index et l'utilisation de l'attribut de valeur concaténée. Puis la section explique le mode de fonctionnement de l'application.
- Étape 3 : déploiement en production à l'aide du service DynamoDB – Cette section se concentre sur les considérations relatives au déploiement en production. Dans cette étape, vous créez une table en utilisant le service Amazon DynamoDB, et déployez l'application à l'aide d'AWS Elastic Beanstalk. Lorsque vous avez l'application en production, vous accordez également les autorisations appropriées afin que l'application puisse accéder à la table DynamoDB. Les instructions de cette section vous guident tout au long du déploiement en end-to-end production.
- Étape 4 : Nettoyer les ressources – Cette section met en évidence les domaines qui ne sont pas couverts par cet exemple. La section indique également les étapes à suivre pour supprimer les AWS ressources que vous avez créées au cours des étapes précédentes afin d'éviter d'encourir des frais.

Étape 1 : déploiement et test localement

Rubriques

- 1.1 : téléchargement et installation des packages requis
- 1.2 : test de l'application de jeu

Au cours de cette étape, vous allez télécharger, déployer et tester l'application de Tic-Tac-Toe jeu sur votre ordinateur local. Au lieu d'utiliser le service web Amazon DynamoDB, vous téléchargez DynamoDB sur votre ordinateur et créez la table obligatoire à cet emplacement.

1.1 : téléchargement et installation des packages requis

Pour tester cette application localement, vous aurez besoin des éléments suivants :

- Python
- Flask (une microstructure pour Python)
- AWS SDK pour Python (Boto)
- DynamoDB s'exécutant sur votre ordinateur
- Git

Pour obtenir ces outils, procédez comme suit :

1. Installez Python. Pour step-by-step obtenir des instructions, voir Télécharger Python.

   L' Tic-Tac-Toeapplication a été testée avec Python version 2.7.
2. Installez Flask et AWS SDK pour Python (Boto) utilisez le Python Package Installer (PIP) :
   - Installez PIP.

     Pour obtenir des instructions, veuillez consulter Installation de PIP. Sur la page de l'installation, choisissez le lien get-pip.py, puis enregistrez le fichier. Ouvrez ensuite un terminal de commande en tant qu'administrateur, puis entrez ce qui suit à l'invite de commande.

     ```
     python.exe get-pip.py
     ```

     Sur Linux, vous ne spécifiez pas l'extension `.exe`. Vous spécifiez uniquement `python get-pip.py`.
   - Avec PIP, installez les packages Flask et Boto en utilisant le code suivant.

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```
3. Téléchargez DynamoDB sur votre ordinateur. Pour plus d'informations sur la façon de l'exécuter, consultez Configuration de DynamoDB Local (version téléchargeable) .

4. Téléchargez l' Tic-Tac-Toeapplication :

   a. Installez Git. Pour obtenir des instructions, consultez Téléchargements git.

   b. Exécutez le code suivant pour télécharger l'application.

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2 : test de l'application de jeu

Pour tester l' Tic-Tac-Toeapplication, vous devez exécuter DynamoDB localement sur votre ordinateur.

Pour exécuter l' tic-tac-toeapplication

1. Démarrez DynamoDB.
2. Démarrez le serveur Web de l' Tic-Tac-Toeapplication.

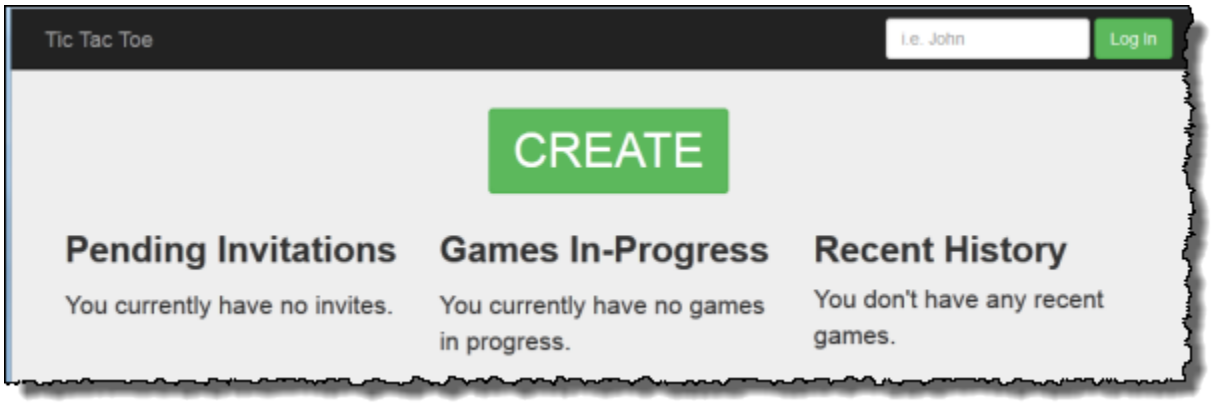
   Pour ce faire, ouvrez un terminal de commande, accédez au dossier dans lequel vous avez téléchargé l' Tic-Tac-Toeapplication et exécutez l'application localement à l'aide du code suivant.

```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

   Sur Linux, vous ne spécifiez pas l'extension `.exe`.
3. Ouvrez votre navigateur web et entrez ce qui suit.

```
http://localhost:5000/
```

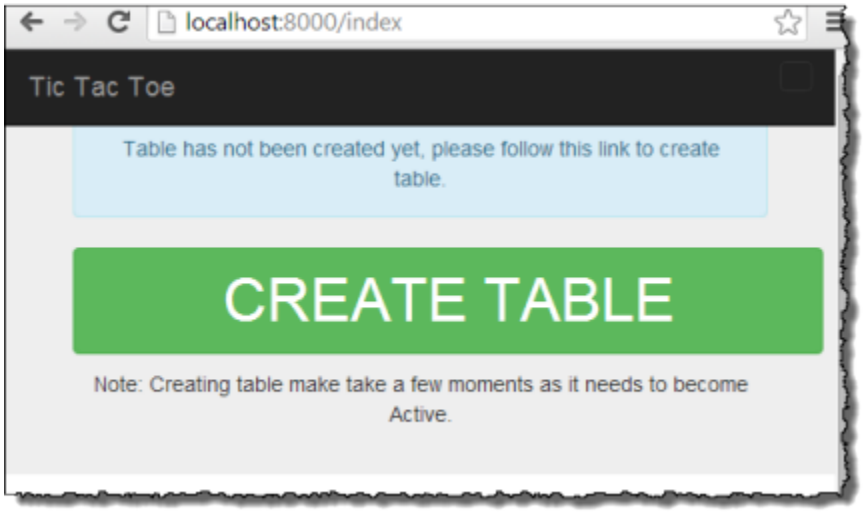
   Le navigateur affiche la page d'accueil.



4. Entrez **`user1`** dans la zone Log in (Connexion) pour vous connecter tant qu'utilisateur user1.

> **Note**
>
> Cet exemple d'application n'effectue aucune authentification de l'utilisateur. L'ID d'utilisateur est utilisé uniquement pour identifier les joueurs. Si deux joueurs se connectent avec le même alias, l'application fonctionne comme si vous jouiez dans deux navigateurs différents.

5. Si vous jouez à ce jeu pour la première fois, une page s'affiche vous demandant de créer la table requise (`Games`) dans DynamoDB. Choisissez CREATE TABLE.



6. Choisissez CRÉER pour créer le premier tic-tac-toe jeu.
7. Entrez **`user2`** dans la zone Choose an Opponent (Choisir un adversaire) et choisissez Create Game! (Créer le jeu !)

Cela crée le jeu en ajoutant un élément dans la table `Games`. Cela définit le statut du jeu sur `PENDING`.

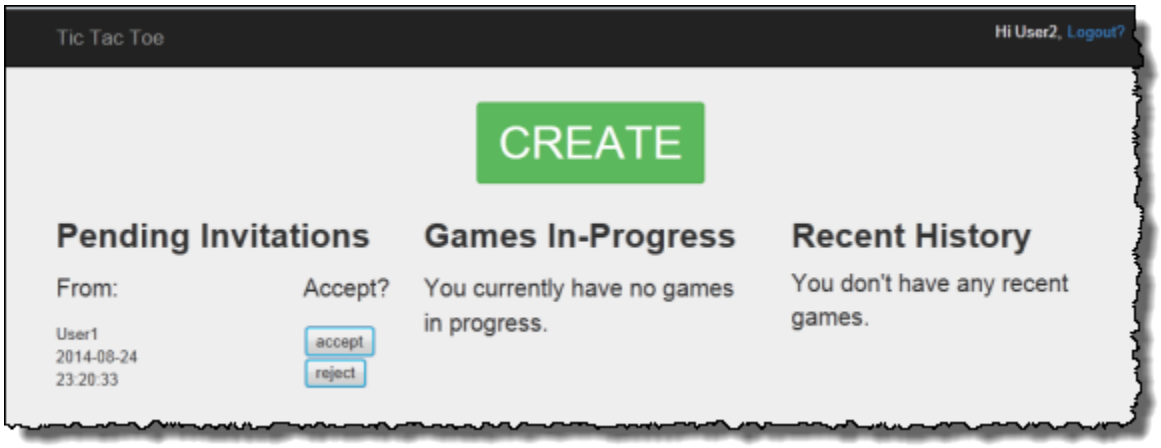
8. Ouvrez une autre fenêtre de navigateur et entrez ce qui suit.

```
http://localhost:5000/
```

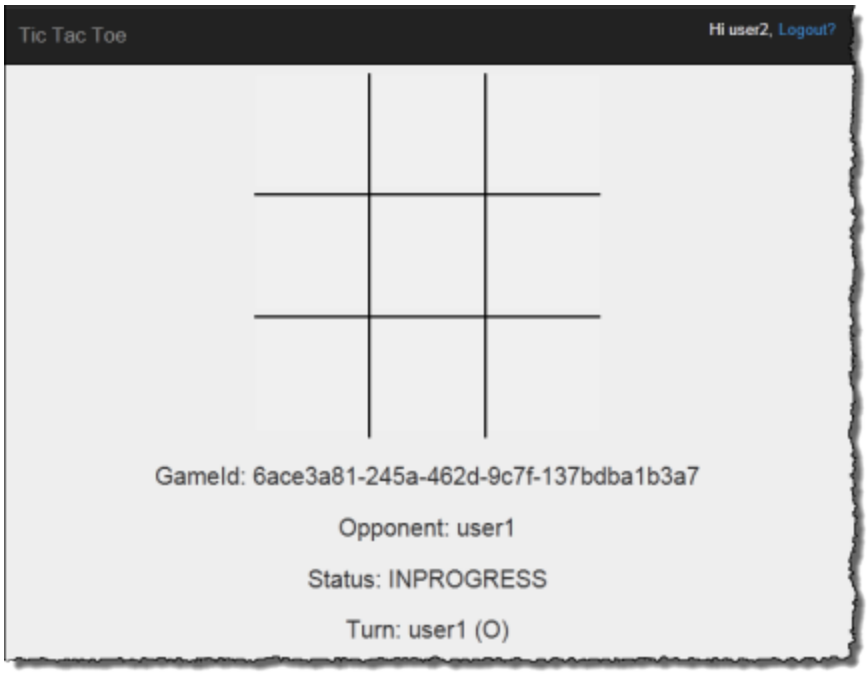
   Le navigateur transmet des informations par le biais de cookies, vous devez donc utiliser le mode de navigation privée afin que vos cookies ne soient pas transférés.

9. Connectez-vous en tant qu'user2.

   Une page s'affiche qui indique une invitation en attente d'user1.



10. Choisissez accept pour accepter l'invitation.



La page du jeu apparaît avec une tic-tac-toe grille vide. La page affiche également des informations de jeu pertinentes telles que l'ID du jeu, le joueur dont c'est le tour et le statut du jeu.

11. Jouez.

Pour chaque mouvement de l'utilisateur, le service web envoie à DynamoDB une demande de mise à jour conditionnelle de l'élément de jeu dans la table `Games`. Par exemple, les conditions garantissent la validité du déplacement, la disponibilité de la case choisie par l'utilisateur et que c'était bien le tour de l'utilisateur qui a effectué le déplacement. Pour un déplacement valide, l'opération de mise à jour ajoute un nouvel attribut correspondant à la sélection sur le plateau. L'opération de mise à jour définit également la valeur de l'attribut existant sur l'utilisateur qui peut procéder au déplacement suivant.

Sur la page du jeu, l'application effectue des JavaScript appels asynchrones toutes les secondes, pendant 5 minutes maximum, pour vérifier si l'état du jeu dans DynamoDB a changé. Si c'est le cas, l'application met à jour la page avec de nouvelles informations. Après 5 minutes, l'application arrête de générer les demandes et vous devez actualiser la page pour obtenir des informations mises à jour.

Étape 2 : examen du modèle de données et des détails de la mise en œuvre

Rubriques

- 2.1 : modèle de données de base
- 2.2 : application en action (procédure pas à pas de code)

2.1 : modèle de données de base

Cet exemple d'application met en évidence les concepts de modèle de données DynamoDB suivants :

- Table – Dans DynamoDB, une table est un ensemble d'éléments (registres), et chaque élément est un ensemble de paires nom-valeur appelées attributs.

  Dans cet Tic-Tac-Toe exemple, l'application stocke toutes les données du jeu dans un tableau,`Games`. L'application crée un élément par jeu dans la table et stocke toutes les données de jeux en tant qu'attributs. Une tic-tac-toe partie peut comporter jusqu'à neuf coups. Les tables

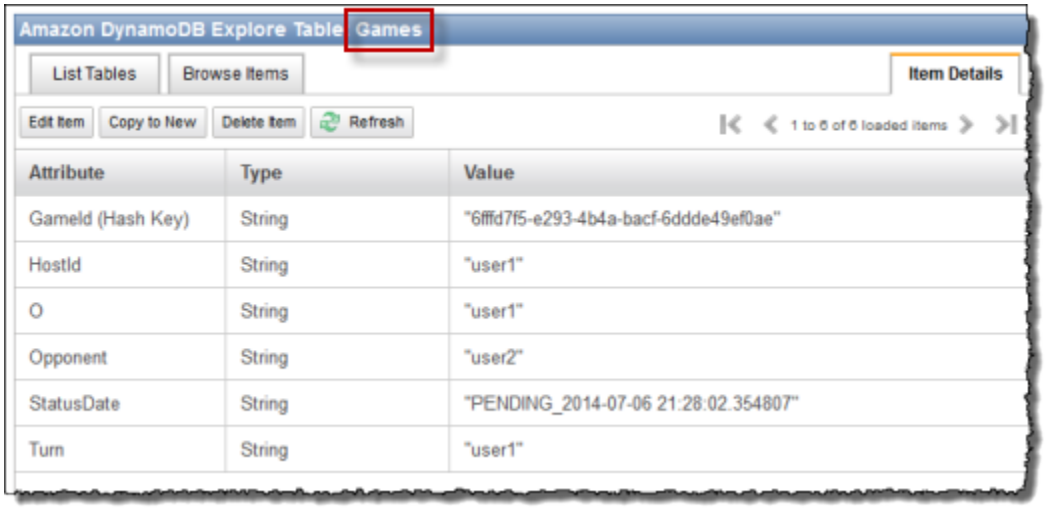
DynamoDB n'ayant pas de schéma dans les cas où seule la clé primaire est l'attribut obligatoire, l'application peut stocker différents nombres d'attributs par élément de jeu.

La table `Games` a une clé primaire simple composée d'un attribut, `GameId`, de type chaîne. L'application attribue un ID unique à chaque partie. Pour plus d'informations sur les clés primaires DynamoDB, consultez Clé primaire.

Lorsqu'un utilisateur lance un tic-tac-toe jeu en invitant un autre utilisateur à jouer, l'application crée un nouvel élément dans le `Games` tableau avec des attributs stockant les métadonnées du jeu, tels que les suivants :

- `HostId`, l'utilisateur qui a lancé la partie.
- `Opponent`, l'utilisateur qui a été invité à jouer.
- L'utilisateur dont c'est le tour de jouer. L'utilisateur qui a lancé la partie joue en premier.
- L'utilisateur qui utilise le symbole O sur le plateau. L'utilisateur qui initie les parties utilise le symbole O.

En outre, l'application crée un attribut concaténé `StatusDate`, marquant l'état du jeu initial en tant que `PENDING`. La capture d'écran suivante présente un exemple d'élément tel qu'il s'affiche dans la console DynamoDB :

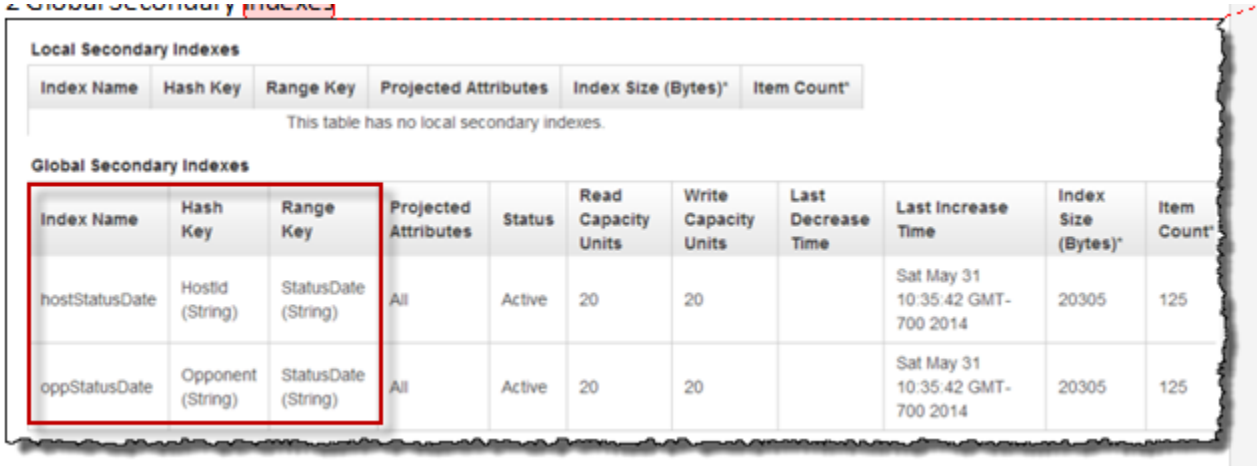


En cours de partie, l'application ajoute un attribut à la table pour chaque déplacement. Le nom d'attribut est la position sur le plateau, par exemple `TopLeft` ou `BottomRight`. Par exemple, un déplacement peut avoir un attribut `TopLeft` avec la valeur `0`, un attribut `TopRight` avec la valeur `0` et un attribut `BottomRight` avec la valeur `X`. La valeur d'attribut est `0` ou `X`, selon l'utilisateur qui a effectué le déplacement. Par exemple, considérez le plateau suivant.

- Attributs de valeur concaténée – L'attribut `StatusDate` illustre un attribut de valeur concaténée. Dans cette approche, au lieu de créer des attributs distincts pour stocker le statut du jeu (`PENDING`, `IN_PROGRESS` et `FINISHED`) et la date (moment du dernier déplacement), vous les associez en un seul attribut, par exemple `IN_PROGRESS_2014-04-30 10:20:32`.

  L'application utilise ensuite l'attribut `StatusDate` pour la création d'index secondaires en spécifiant `StatusDate` comme clé de tri pour l'index. L'avantage d'utiliser l'attribut de valeur concaténée `StatusDate` est illustré plus en détails dans les index abordés par la suite.

- Index secondaires globaux – Vous pouvez utiliser la clé primaire de la table, `GameId`, pour interroger efficacement la table afin de trouver un élément de jeu. Pour interroger la table sur des attributs autres que les attributs de clé primaires, DynamoDB prend en charge la création d'index secondaires. Dans cet exemple d'application, vous créez les deux index secondaires suivants :



Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*

This table has no local secondary indexes.

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

  - HostId- StatusDate -indice. Cet index a `HostId` comme clé de partition et `StatusDate` comme une clé de tri. Vous pouvez utiliser cet index pour interroger sur `HostId`, par exemple pour trouver des jeux hébergés par un utilisateur particulier.

- OpponentId- StatusDate -indice. Cet index a `OpponentId` comme clé de partition et `StatusDate` comme une clé de tri. Vous pouvez utiliser cet index pour interroger sur `Opponent`, par exemple pour trouver des jeux dans lesquels un utilisateur particulier est l'adversaire.

Ces index sont appelés index secondaires globaux, car la clé de partition dans ces index n'est pas la même que la clé de partition (`GameId`) utilisée dans la clé primaire de la table.

Notez que les deux index spécifient `StatusDate` comme clé de tri. Cela permet d'effectuer les opérations suivantes :

- Vous pouvez interroger à l'aide de l'opérateur de comparaison `BEGINS_WITH`. Par exemple, vous pouvez trouver tous les jeux avec l'attribut `IN_PROGRESS` hébergés par un utilisateur particulier. Dans ce cas, l'opérateur `BEGINS_WITH` vérifie la valeur `StatusDate` qui commence par `IN_PROGRESS`.
- DynamoDB stocke les éléments dans l'index dans l'ordre, par valeur de clé de tri. Donc, si tous les préfixes de statut sont identiques (par exemple, `IN_PROGRESS`), le format ISO utilisé pour la partie date aura des éléments triés du plus ancien au plus récent. Cette approche permet d'effectuer certaines requêtes efficacement, par exemple les éléments suivants :
  - Récupérez jusqu'à 10 des jeux `IN_PROGRESS` les plus récents organisés par l'utilisateur qui est connecté. Pour cette requête, vous spécifiez l'index `HostId-StatusDate-index`.
  - Récupérez jusqu'à 10 des jeux `IN_PROGRESS` les plus récents où l'utilisateur connecté est l'adversaire. Pour cette requête, vous spécifiez l'index `OpponentId-StatusDate-index`.

Pour plus d'informations sur le fonctionnement des index secondaires, consultez Amélioration de l'accès aux données avec les index secondaires dans DynamoDB.

2.2 : application en action (procédure pas à pas de code)

Cette application a deux pages principales :

- Page d'accueil — Cette page fournit à l'utilisateur un identifiant simple, un bouton CRÉER pour créer un nouveau tic-tac-toe jeu, une liste des parties en cours, un historique des parties et toutes les invitations à des jeux en attente.

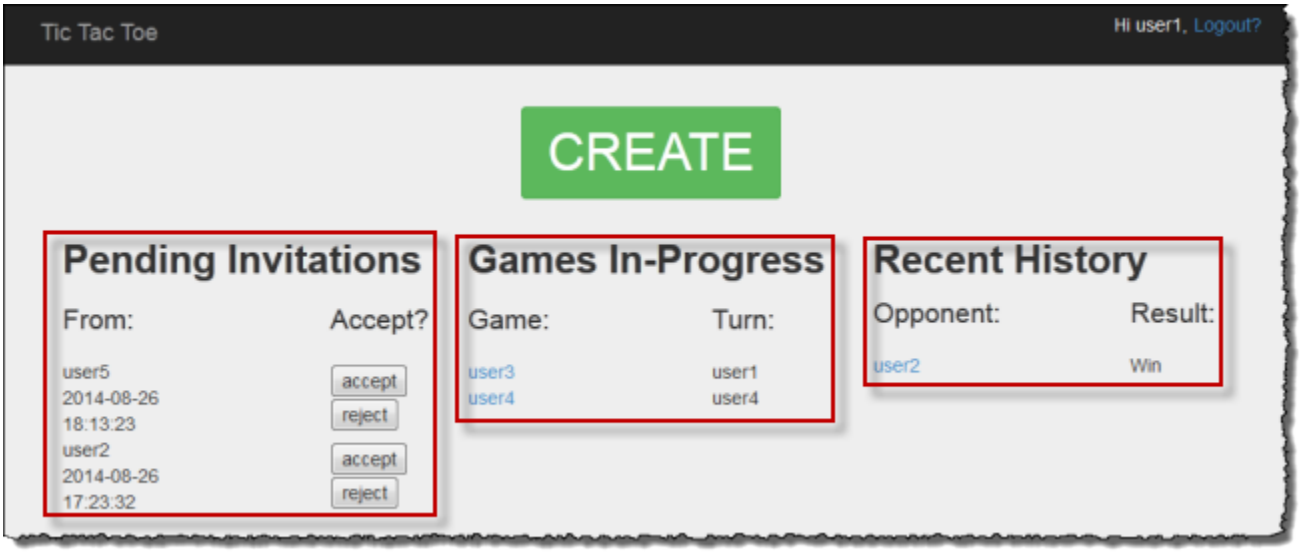
  La page d'accueil n'est pas actualisée automatiquement. Vous devez actualiser la page pour actualiser les listes.
- Page de jeu — Cette page montre la tic-tac-toe grille sur laquelle les utilisateurs jouent.

L'application met à jour la page de jeu automatiquement chaque seconde. JavaScript Dans votre navigateur, le serveur Web Python appelle le serveur Web Python toutes les secondes pour demander à la table Games si les éléments du jeu de la table ont changé. Si tel est le cas, JavaScript déclenche une actualisation de la page afin que l'utilisateur puisse voir le tableau mis à jour.

Nous verrons en détails le mode de fonctionnement de l'application.

Page d'accueil

Une fois l'utilisateur connecté, l'application affiche les trois listes d'informations suivantes :



- Invitations – Cette liste affiche jusqu'aux 10 dernières invitations d'autres personnes, qui sont en attente d'acceptation par l'utilisateur connecté. Dans la capture d'écran précédente, user1 dispose d'invitations d'user5 et d'user2 en attente.
- Games in-progress – Cette liste affiche jusqu'aux 10 dernière parties en cours. Il s'agit de parties auxquelles l'utilisateur participe activement et qui ont le statut `IN_PROGRESS`. Sur la capture d'écran, l'utilisateur1 joue activement à un tic-tac-toe jeu avec utilisateur3 et utilisateur4.
- Recent history – Cette liste affiche jusqu'aux 10 dernières parties que l'utilisateur a terminées, dont le statut est `FINISHED`. Dans la partie illustrée dans la capture d'écran, user1 a joué avec user2. Pour chaque partie terminée, la liste présente le résultat.

Dans le code, la fonction `index` (dans `application.py`) effectue les trois appels suivants pour récupérer des informations sur le statut du jeu :

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Chacun de ces appels renvoie une liste d'éléments de DynamoDB qui sont encapsulés par les objets `Game`. Il est facile d'extraire des données de ces objets dans la vue. La fonction index passe ces listes d'objets à la vue pour afficher le HTML.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

L' Tic-Tac-Toeapplication définit la `Game` classe principalement pour stocker les données de jeu extraites de DynamoDB. Ces fonctions renvoient des listes d'objets `Game` qui vous permettent d'isoler le reste de l'application du code relatif aux éléments Amazon DynamoDB. Par conséquent, ces fonctions vous aident à découpler le code de votre application à partir des détails de la couche de stockage de données.

Le modèle architectural décrit ici est également appelé modèle d'interface utilisateur model-view-controller (MVC). Dans ce cas, les instances d'objet `Game` (qui représentent des données) sont le modèle et la page HTML est la vue. Le contrôleur est divisé en deux fichiers. Le fichier `application.py` a la logique de contrôleur pour la structure Flask, et la logique métier est isolée dans le fichier `gameController.py`. Autrement dit, l'application stocke tout ce qui est associé au kit SDK DynamoDB dans son propre fichier dans le dossier `dynamodb`.

Permettez-nous de passer en revue les trois fonctions et comment elles interrogent la table Games à l'aide des index secondaires globaux pour récupérer les données concernés.

Utilisation getGameInvites pour obtenir la liste des invitations à des jeux en attente

La fonction `getGameInvites` récupère la liste des 10 dernières invitations en suspens. Ces jeux ont été créés par des utilisateurs, mais les adversaires n'ont pas accepté les invitations à jouer. Pour ces jeux, l'état reste `PENDING` jusqu'à ce que l'adversaire accepte l'invitation. Si l'adversaire décline l'invitation, l'application supprime l'élément correspondant de la table.

La fonction spécifie la requête comme suit :

- Elle spécifie l'index `OpponentId-StatusDate-index` à utiliser avec les valeurs de clés d'index et les opérateurs de comparaison suivants :
  - La clé de partition est `OpponentId` et prend la clé d'index *`user ID`*.
  - La clé de tri est `StatusDate` et prend l'opérateur de comparaison et la valeur de clé d'index `beginswith="PENDING_"`.

  Vous utilisez l'index `OpponentId-StatusDate-index` pour récupérer les jeux auxquels l'utilisateur connecté est invité, c'est-à-dire où l'utilisateur connecté est l'adversaire.
- La requête limite le résultat à 10 éléments.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

Dans l'index, pour chaque `OpponentId` (clé de partition), DynamoDB conserve les éléments triés par `StatusDate` (clé de tri). Par conséquent, les parties que la requête renvoie seront les 10 parties les plus récentes.

Utiliser getGamesWith le statut pour obtenir la liste des jeux ayant un statut spécifique

Une fois qu'un adversaire accepte une invitation, le statut du jeu passe sur `IN_PROGRESS`. Une fois que le jeu se termine, le statut passe sur `FINISHED`.

Les requêtes pour trouver des jeux qui sont en cours ou terminés sont identiques, à l'exception de la valeur du statut qui est différente. Par conséquent, l'application définit la fonction `getGamesWithStatus`, qui prend la valeur de statut comme paramètre.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

La section suivante décrit les jeux en cours, mais la même description s'applique également aux jeux terminés.

Une liste de jeux en cours pour un utilisateur donné comprend les deux éléments suivants :

- Jeux en cours hébergés par l'utilisateur

- Jeux en cours où l'utilisateur est l'adversaire

La fonction `getGamesWithStatus` exécute les deux requêtes suivantes, chaque fois à l'aide de l'index secondaire approprié.

- La fonction interroge la table `Games` en utilisant l'index `HostId-StatusDate-index`. Pour l'index, la requête spécifie les valeurs de clés primaires, clé de partition (`HostId`) et clé de tri (`StatusDate`), ainsi que les opérateurs de comparaison.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                       StatusDate__beginswith=status,
                                                       index="HostId-StatusDate-index",
                                                       limit=10)
  ```

  Notez la syntaxe Python pour les opérateurs de comparaison :

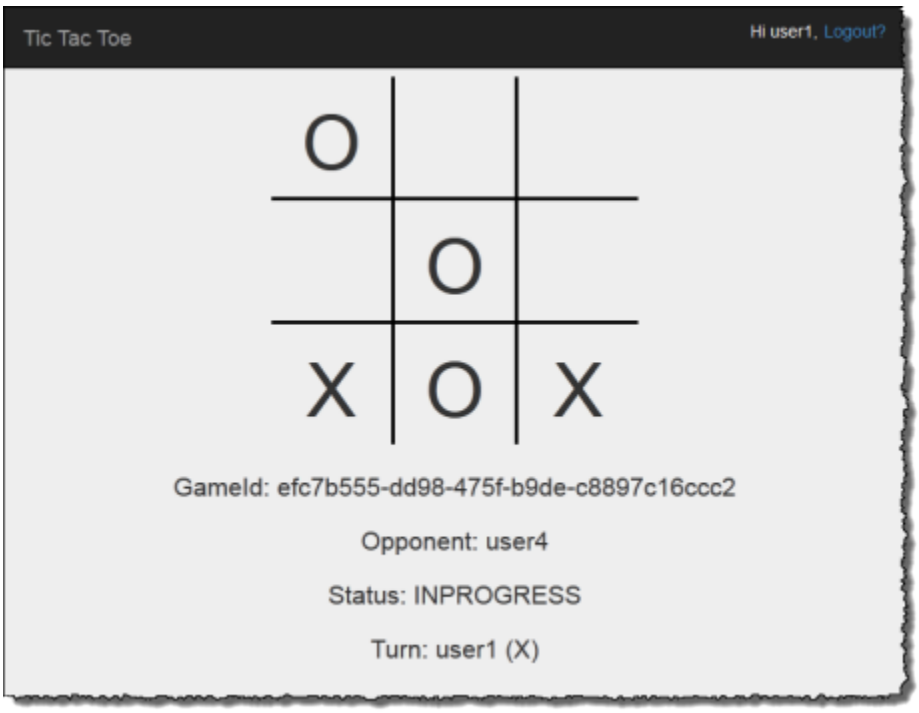
  - `HostId__eq=user` spécifie l'opérateur de comparaison d'égalité.
  - `StatusDate__beginswith=status` spécifie l'opérateur de comparaison `BEGINS_WITH`.
- La fonction interroge la table `Games` en utilisant l'index `OpponentId-StatusDate-index`.

  ```
  oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="OpponentId-StatusDate-index",
                                                     limit=10)
  ```
- Ensuite, la fonction combine les deux listes, trie et pour les premiers 0 à 10 éléments, elle crée une liste des objets `Game` et renvoie la liste à la fonction d'appel (à savoir, l'index).

  ```
  games = self.mergeQueries(hostGamesInProgress,
                            oppGamesInProgress)
  return games
  ```

Page de jeu

La page de jeu est l'endroit où l'utilisateur joue à tic-tac-toe des jeux. Elle présente la grille de jeu ainsi que des informations relatives au jeu. La capture d'écran suivante illustre un exemple de jeu en cours :



L'application affiche la page de jeu dans les cas suivants :

- L'utilisateur crée un jeu en invitant un autre utilisateur à jouer.

  Dans ce cas, la page affiche l'utilisateur comme hôte et le statut de jeu comme `PENDING`, en attendant que l'adversaire accepte.
- L'utilisateur accepte l'une des invitations en attente sur la page d'accueil.

  Dans ce cas, la page affiche l'utilisateur comme adversaire et le statut du jeu comme `IN_PROGRESS`.

Une sélection d'utilisateur sur le plateau génère une demande `POST` de formulaire à l'application. Autrement dit, Flask appelle la fonction `selectSquare` (dans `application.py`) avec les données de formulaire HTML. Cette fonction, à son tour, appelle la fonction `updateBoardAndTurn` (dans `gameController.py`) pour mettre à jour l'élément de jeu comme suit :

- Elle ajoute un nouvel attribut spécifique au déplacement.
- Elle met à jour la valeur d'attribut `Turn` pour l'utilisateur qui jouera le prochain coup.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

La fonction renvoie la valeur true si la mise à jour de l'élément a réussi ; sinon, elle renvoie la valeur false. Notez les éléments suivants sur la fonction `updateBoardAndTurn` :

- La fonction appelle la fonction `update_item` du kit SDK pour Python afin d'apporter un ensemble fini de mises à jour à un élément existant. La fonction mappe à l'opération `UpdateItem` dans DynamoDB. Pour de plus amples informations, veuillez consulter UpdateItem.

> **Note**
>
> La différence entre les opérations `UpdateItem`et `PutItem` est que `PutItem` remplace l'élément entier. Pour de plus amples informations, veuillez consulter PutItem.

Pour l'appel `update_item`, le code identifie les éléments suivants :

- La clé primaire de la table `Games` (à savoir, `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- Le nouvel attribut à ajouter, spécifique au déplacement de l'utilisateur en cours et sa valeur (par exemple, `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Conditions qui doivent avoir une valeur true pour que la mise à jour ait lieu :
  - La partie doit être en cours. C'est à dire que la valeur d'attribut `StatusDate` doit commencer par `IN_PROGRESS`.
  - Le tour actuel doit être un tour d'utilisateur valide comme indiqué par l'attribut `Turn`.
  - L'emplacement que l'utilisateur choisit doit être disponible. Autrement dit, l'attribut correspondant à l'emplacement ne doit pas exister.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
```

```
    position : {"Exists" : False}}
```

Maintenant, la fonction appelle `update_item` pour mettre à jour l'élément.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Après le renvoi de la fonction, les appels de la fonction `selectSquare` sont redirigés, comme illustré dans l'exemple suivant.

```
redirect("/game="+gameId)
```

Cet appel permet au navigateur de s'actualiser. Dans le cadre de cette actualisation, l'application vérifie si la partie s'est terminée par une victoire ou un nul. Si c'est le cas, l'application met à jour l'élément de partie en conséquence.

Étape 3 : déploiement en production à l'aide du service DynamoDB

Rubriques

- 3.1 : Création d'un rôle IAM pour Amazon EC2
- 3.2 : création de la table Games dans Amazon DynamoDB
- 3.3 : Regroupez et déployez le code de tic-tac-toe l'application
- 3.4 : configuration de l'environnement AWS Elastic Beanstalk

Dans les sections précédentes, vous avez déployé et testé l' Tic-Tac-Toeapplication localement sur votre ordinateur à l'aide de DynamoDB local. Maintenant, vous déployez l'application en production comme suit :

- Déployez l'application àAWS Elastic Beanstalk l'aide d'un easy-to-use service de déploiement et de dimensionnement d'applications et de services Web. Pour plus d'informations, consultez la section Déploiement d'une application Flask vers AWS Elastic Beanstalk.

  Elastic Beanstalk lance une ou plusieurs instances Amazon Elastic Compute Cloud (EC2Amazon), que vous configurez via Elastic Beanstalk, sur lesquelles votre application sera exécutée. Tic-Tac-Toe

- À l’aide du service Amazon DynamoDB, créez une table `Games` qui résidera sur AWS plutôt que localement sur votre ordinateur.

En outre, vous devez également configurer des autorisations. Toutes AWS les ressources que vous créez, telles que la `Games` table dans DynamoDB, sont privées par défaut. Seul le propriétaire des ressources, à savoir le compte AWS qui a créé la table `Games`, peut accéder à cette table. Ainsi, par défaut, votre Tic-Tac-Toe application ne peut pas mettre à jour le `Games` tableau.

Pour accorder les autorisations nécessaires, vous créez un rôle Gestion des identités et des accès AWS(IAM) et accordez à ce rôle les autorisations d'accès à la `Games` table. Votre EC2 instance Amazon assume d'abord ce rôle. En réponse,AWS renvoie des informations de sécurité temporaires que l' EC2 instance Amazon peut utiliser pour mettre à jour le `Games` tableau au nom de l' Tic-Tac-Toeapplication. Lorsque vous configurez votre application Elastic Beanstalk, vous spécifiez le rôle IAM que l'instance ou les instances EC2 Amazon peuvent assumer. Pour plus d'informations sur les rôles IAM, consultez la section Rôles IAM pour Amazon EC2 dans le guide de EC2 l'utilisateur Amazon.

> **Note**
>
> Avant de créer des EC2 instances Amazon pour l' Tic-Tac-Toeapplication, vous devez d'abord choisir la AWS région dans laquelle vous souhaitez qu'Elastic Beanstalk crée les instances. Une fois que vous avez créé l’application Elastic Beanstalk, vous fournissez les mêmes point de terminaison et nom de région dans un fichier de configuration. L' Tic-Tac-Toeapplication utilise les informations contenues dans ce fichier pour créer la `Games` table et envoyer les demandes suivantes dans une AWS région spécifique. La table `Games` DynamoDB et les instances EC2 Amazon lancées par Elastic Beanstalk doivent se trouver dans la même région. Pour obtenir la liste des régions disponibles, veuillez consulter Amazon DynamoDB dans le Référence générale d'Amazon Web Services.

En résumé, vous devez effectuer les opérations suivantes pour déployer l' Tic-Tac-Toeapplication en production :

1. Créez un rôle IAM à l’aide du service IAM. Vous attachez à ce rôle une politique accordant à des actions DynamoDB des autorisations pour accéder à la table `Games`.
2. Regroupez le code de Tic-Tac-Toe l'application et un fichier de configuration, puis créez un `.zip` fichier. Vous utilisez ce `.zip` fichier pour transmettre le code de l' Tic-Tac-Toeapplication à Elastic

Beanstalk afin qu'il puisse l'installer sur vos serveurs. Pour plus d'informations sur la création d'une solution groupée, consultez Création d'une solution groupée d'application dans le Guide du développeur AWS Elastic Beanstalk.

Dans le fichier de configuration (`beanstalk.config`), vous fournissez des informations de point de terminaison et de région AWS. L' Tic-Tac-Toeapplication utilise ces informations pour déterminer à quelle région DynamoDB communiquer.

3. Configurez l'environnement Elastic Beanstalk. Elastic Beanstalk lance EC2 une ou plusieurs instances Amazon Tic-Tac-Toe et y déploie votre bundle d'applications. Une fois que l'environnement Elastic Beanstalk est prêt, vous fournissez le nom du fichier de configuration en ajoutant la variable d'environnement `CONFIG_FILE`.
4. Créez la table DynamoDB À l'aide du service Amazon DynamoDB, vous créez `Games` la table AWS sur votre ordinateur plutôt que localement. N'oubliez pas que cette table a une clé primaire simple constituée de la clé de partition `GameId` de type chaîne.
5. Testez le jeu en production.

3.1 : Création d'un rôle IAM pour Amazon EC2

La création d'un rôle IAM de EC2 type Amazon permet à l' EC2instance Amazon qui exécute votre Tic-Tac-Toe application d'assumer le rôle approprié et de demander à l'application d'accéder à la `Games` table. Lorsque vous créez le rôle, sélectionnez l'option Politique personalisée puis copiez et collez la politique suivante.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
         "Action":[
            "dynamodb:*"
```

```
            ],
            "Effect":"Allow",
            "Resource":[
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
            ]
         }
      ]
}
```

Pour plus d'instructions, consultez Création d'un rôle pour un service AWS(AWS Management Console) dans le Guide de l'utilisateur IAM.

3.2 : création de la table Games dans Amazon DynamoDB

La table `Games` dans DynamoDB stocke les données des parties. Si la table n'existe pas, l'application la crée pour vous. Dans ce cas, laissez l'application créer la table `Games`.

3.3 : Regroupez et déployez le code de tic-tac-toe l'application

Si vous avez suivi les étapes de cet exemple, l' Tic-Tac-Toeapplication est déjà téléchargée. Dans le cas contraire, téléchargez l'application et extrayez tous les fichiers dans un dossier sur votre ordinateur local. Pour obtenir des instructions, veuillez consulter Étape 1 : déploiement et test localement.

Une fois tous les fichiers extraits, vous disposez d'un dossier `code`. Pour remettre ce dossier à Elastic Beanstalk, vous regroupez le contenu de ce dossier en tant que fichier `.zip`. Tout d'abord, vous ajoutez un fichier de configuration dans ce dossier. Votre application utilise les informations de région et de point de terminaison pour créer une table DynamoDB dans la région spécifiée, et effectuer les demandes d'opération de table subséquentes à l'aide du point de terminaison spécifié.

1. Accédez au dossier dans lequel vous avez téléchargé l' Tic-Tac-Toeapplication.
2. Dans le dossier racine de l'application, créez un fichier texte nommé `beanstalk.config` avec le contenu suivant.

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   Par exemple, vous pouvez utiliser le contenu suivant.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

Pour obtenir la liste des régions disponibles, consultez Amazon DynamoDB dans la Référence générale d'Amazon Web Services.

> **Important**
>
> La région spécifiée dans le fichier de configuration est l'emplacement où l' Tic-Tac-Toeapplication crée la `Games` table dans DynamoDB. Vous devez créer l'application Elastic Beanstalk abordée dans la section suivante dans la même région.

> **Note**
>
> Lorsque vous créez votre application Elastic Beanstalk, vous demandez à lancer un environnement dans lequel vous pouvez choisir le type d'environnement. Pour tester l' Tic-Tac-Toeexemple d'application, vous pouvez choisir le type d'environnement à instance unique, ignorer les étapes suivantes et passer à l'étape suivante. Toutefois, le type d'environnement Équilibrage de charge, Auto Scaling fournit un environnement hautement disponible et dimensionnable, élément que vous devriez prendre en compte lorsque vous créez et déployez d'autres applications. Si vous choisissez ce type d'environnement, vous devez également générer un UUID et l'ajouter au fichier de configuration, comme illustré ci-après.
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> Dans la communication client-serveur lorsque le serveur envoie une réponse, par souci de sécurité le serveur envoie un cookie signé que le client renvoie au serveur dans la demande suivante. Lorsqu'il y a un seul serveur, le serveur peut générer localement une clé de chiffrement lorsqu'il démarre. Lorsqu'il existe de nombreux serveurs, ils ont tous besoin de connaître la même clé de chiffrement. Dans le cas contraire, ils ne pourront

lire les cookies définis par les serveurs pairs. En ajoutant `secret_key` dans le fichier de configuration, vous indiquez à tous les serveurs d'utiliser cette clé de chiffrement.

3. Zippez le contenu du dossier racine de l'application (qui inclut le fichier `beanstalk.config`), par exemple, `TicTacToe.zip`.
4. Chargez le fichier `.zip` dans un compartiment Amazon Simple Storage Service (Amazon S3). Dans la section suivante, vous allez fournir ce fichier `.zip` à Elastic Beanstalk pour qu'il le charge sur le ou les serveurs.

   Pour des instructions sur la manière de charger un fichier dans un compartiment Amazon S3, consultez Créer un compartiment et Ajouter un objet à un compartiment dans le Guide de l'utilisateur d'Amazon Simple Storage Service.

3.4 : configuration de l'environnement AWS Elastic Beanstalk

Dans cette étape, vous allez créer une application Elastic Beanstalk, qui est un ensemble de composants incluant des environnements. Dans cet exemple, vous lancez une EC2 instance Amazon pour déployer et exécuter votre Tic-Tac-Toe application.

1. Entrez l'URL personnalisée suivante afin de configurer une console Elastic Beanstalk pour configurer l'environnement.

   ```
   https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
   newApplication
   ?applicationName=TicTacToeyour-name
   &solutionStackName=Python
   &sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
   &environmentType=SingleInstance
   &instanceType=t1.micro
   ```

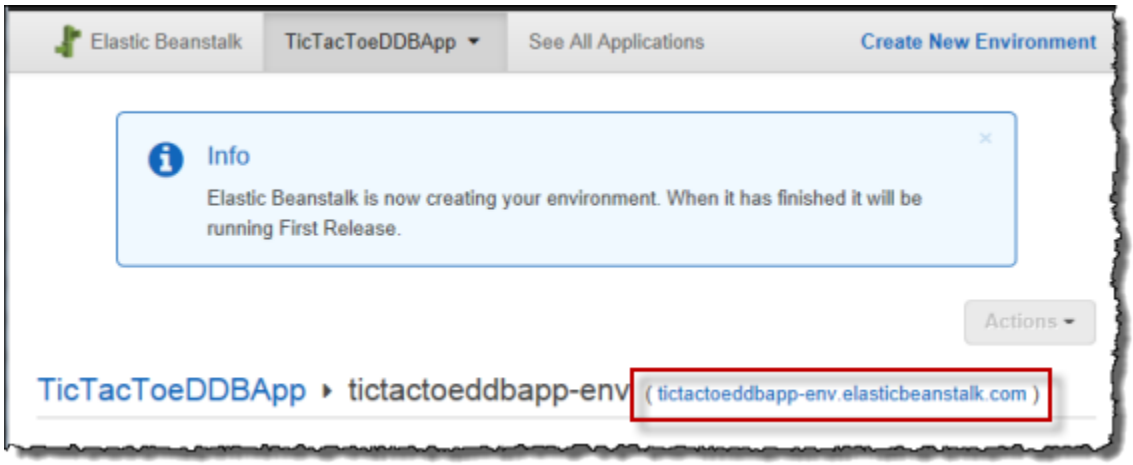
   Pour plus d'informations sur la personnalisation URLs, consultez la section Création d'une URL Launch Now dans le manuel du AWS Elastic Beanstalk développeur. Pour l'URL, notez les points suivants :

   - Vous devez fournir un nom de AWS région (identique à celui que vous avez fourni dans le fichier de configuration), un nom de compartiment Amazon S3 et le nom de l'objet.
   - Pour les tests, l'URL demande le type d'SingleInstanceenvironnement et `t1.micro` en tant que type d'instance.

- Le nom de l'application doit être unique. Ainsi, dans l'URL précédente, nous vous proposons d'ajouter votre nom au `applicationName`.

Cela a pour effet d'ouvrir la console Elastic Beanstalk. Dans certains cas, vous devrez peut-être vous connecter.

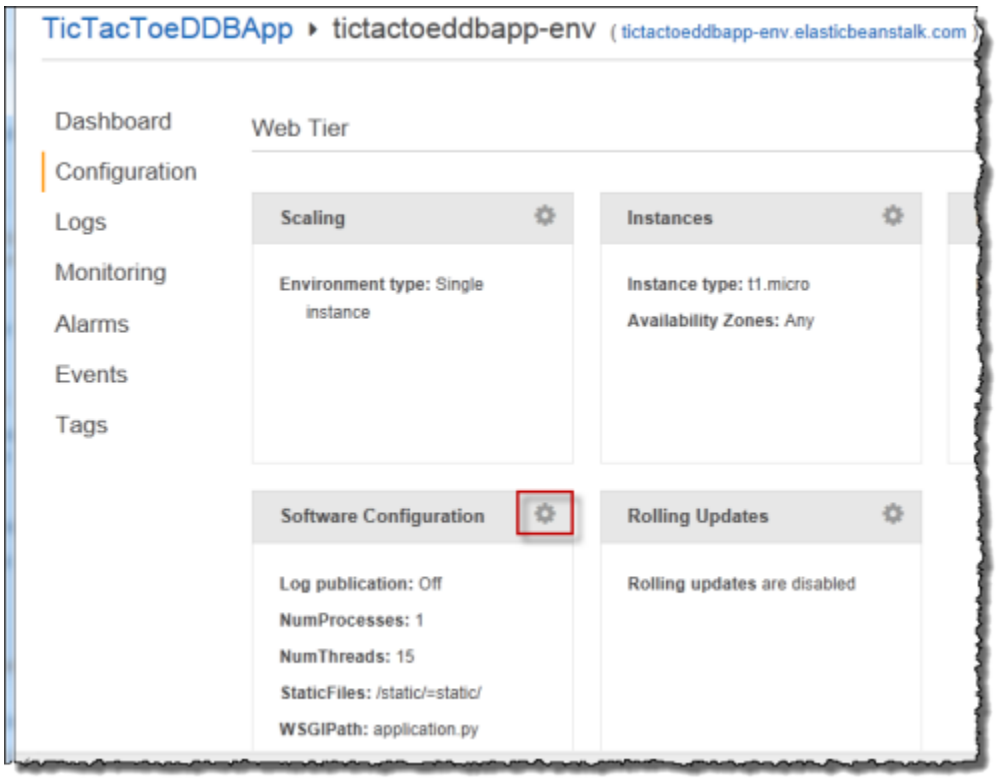
2. Sur la console Elastic Beanstalk, choisissez Vérifier et lancer, puis Lancer.
3. Notez l'URL ultérieurement. Cette URL ouvre la page Tic-Tac-Toe d'accueil de votre application.



4. Configurez l' Tic-Tac-Toeapplication afin qu'elle connaisse l'emplacement du fichier de configuration.

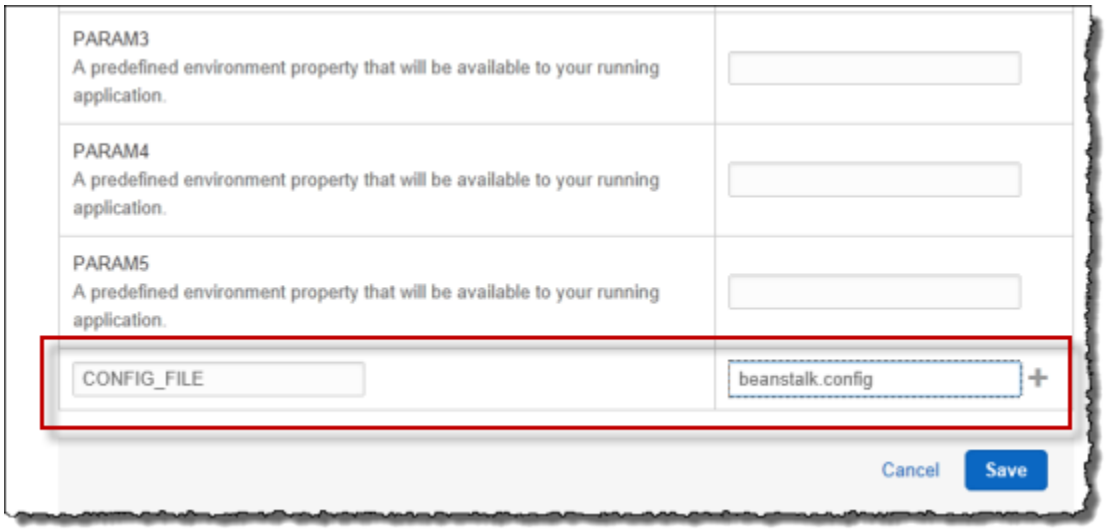
   Une fois qu'Elastic Beanstalk a créé l'application, choisissez Configuration.

   a. Choisissez l'icône de roue dentée à côté de Software Configuration (Configuration logicielle), comme illustré dans la capture d'écran suivante.



b. À la fin de la section Propriétés de l'environnement, entrez **`CONFIG_FILE`** et sa valeur **`beanstalk.config`**, puis choisissez Enregistrer.

Cette mise à jour de l'environnement peut prendre quelques minutes.

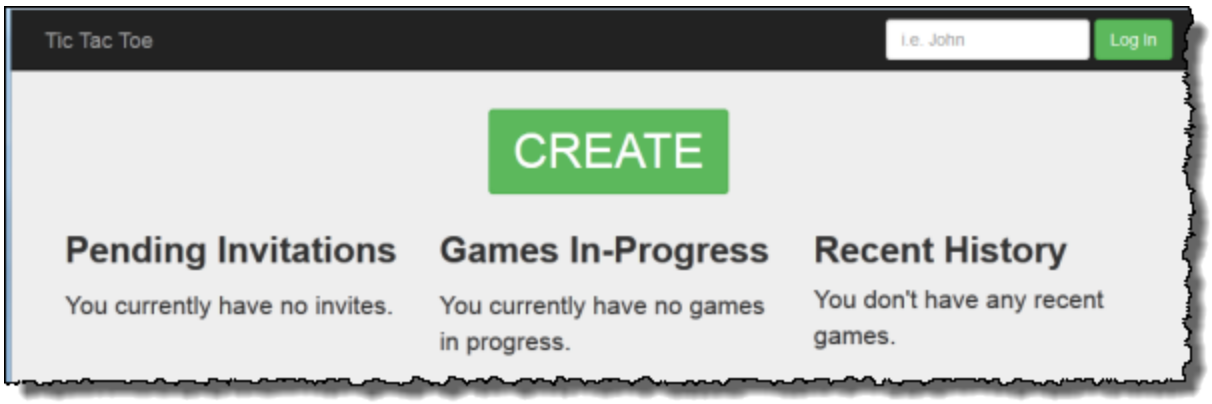


Une fois la mise à jour terminée, vous pouvez jouer au jeu.

5. Dans le navigateur, entrez l'URL que vous avez copiée à l'étape précédente, comme illustré dans l'exemple suivant.

```
http://<pen-name>.elasticbeanstalk.com
```

Cela ouvre la page d'accueil de l'application.



6. Connectez-vous en tant que testuser1, puis choisissez CREATE pour démarrer une nouvelle tic-tac-toe partie.
7. Entrez **`testuser2`** dans la zone Choose an Opponent (Choisir un adversaire).

8. Ouvrez une autre fenêtre de navigateur.

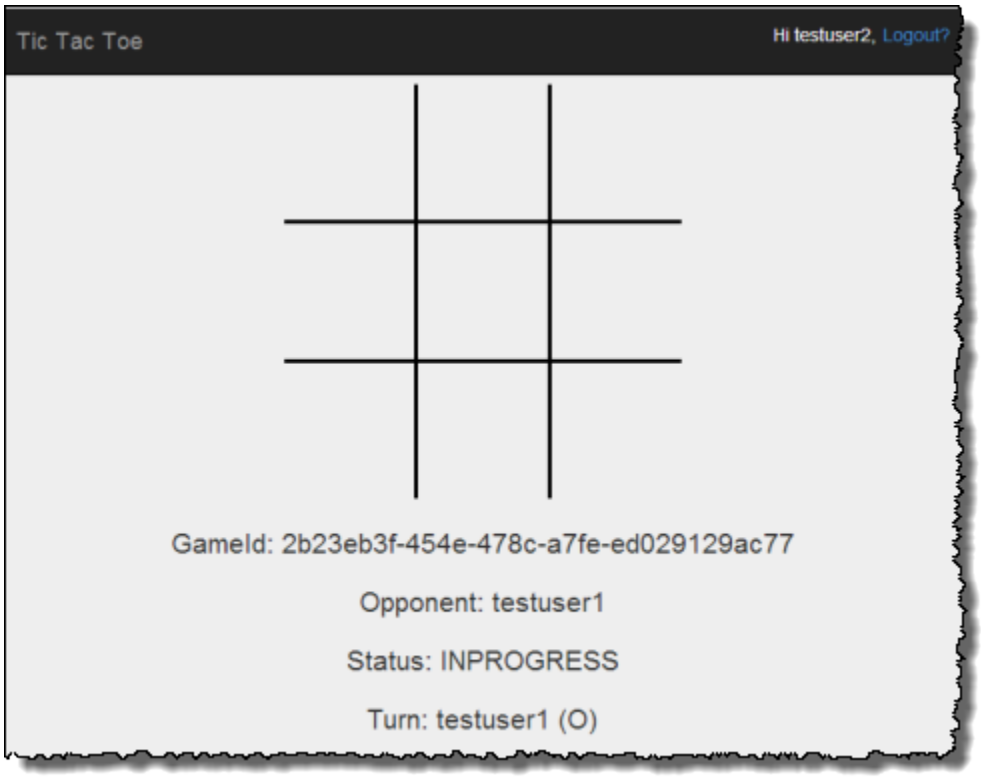
   Assurez-vous de désactiver tous les cookies dans votre fenêtre de navigateur afin de ne pas être connecté en tant que même utilisateur.
9. Entrez la même URL pour ouvrir la page d'accueil de l'application, comme illustré dans l'exemple suivant.

```
http://<env-name>.elasticbeanstalk.com
```

10. Connectez-vous en tant que testuser2.

11. Pour l'invitation de testuser1 dans la liste des invitations en suspens, choisissez accept.

12. À présent, la page de jeu apparaît.



testuser1 et testuser2 peuvent jouer au jeu. Pour chaque déplacement, l'application enregistre le déplacement dans l'élément correspondant de la table `Games`.

Étape 4 : Nettoyer les ressources

Vous avez maintenant terminé le déploiement et les tests de Tic-Tac-Toe l'application. L'application couvre le développement d'applications end-to-end Web sur Amazon DynamoDB, à l'exception de l'authentification des utilisateurs. L'application utilise les informations de connexion sur la page d'accueil uniquement pour ajouter un nom de joueur lors de la création d'un jeu. Dans une application de production, vous ajoutez le code nécessaire pour effectuer l'authentification et la connexion de l'utilisateur.

Si vous avez terminé les tests, vous pouvez supprimer les ressources que vous avez créées pour tester l' Tic-Tac-Toeapplication afin d'éviter d'encourir des frais.

Pour supprimer les ressources que vous avez créées

1. Supprimez la table `Games` que vous avez créée dans DynamoDB.
2. Mettez fin à l'environnement Elastic Beanstalk pour libérer les instances Amazon. EC2
3. Supprimez le rôle IAM que vous avez créé.
4. Supprimez l'objet que vous avez créé dans Amazon S3.

Mots réservés dans DynamoDB

Les mots clés suivants sont réservés à l'usage de DynamoDB. N'utilisez aucun de ces mots comme nom d'attribut dans des expressions. Cette liste n'est pas sensible à la casse.

Si vous avez besoin d'écrire une expression contenant un nom d'attribut en conflit avec un mot réservé dans DynamoDB, vous pouvez définir un nom d'attribut d'expression à utiliser à la place de celui-ci. Pour de plus amples informations, consultez Noms d'attributs d'expression (alias) dans DynamoDB.

```
ABORT
ABSOLUTE
ACTION
ADD
AFTER
AGENT
AGGREGATE
ALL
ALLOCATE
```

```
ALTER
ANALYZE
AND
ANY
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
BULK
BY
BYTE
CALL
CALLED
CALLING
```

```
CAPACITY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
COUNTER
```

```
CREATE
CROSS
CUBE
CURRENT
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
EACH
```

```
ELEMENT
ELSE
ELSEIF
EMPTY
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
FINISH
FIRST
FIXED
FLATTERN
FLOAT
FOR
```

```
FORCE
FOREIGN
FORMAT
FORWARD
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
INDICATOR
```

```
INFINITE
INITIALLY
INLINE
INNER
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
LIMITED
LINES
LIST
LOAD
LOCAL
LOCALTIME
```

```
LOCALTIMESTAMP
LOCATION
LOCATOR
LOCK
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NOT
NULL
NULLIF
```

```
NUMBER
NUMERIC
OBJECT
OF
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
POSITION
```

```
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
REQUEST
```

```
RESET
RESIGNAL
RESOURCE
RESPONSE
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
SET
```

```
SETS
SHARD
SHARE
SHARED
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
SUPER
```

```
SYMMETRIC
SYNONYM
SYSTEM
TABLE
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
UNTIL
```

```
UPDATE
UPPER
URL
USAGE
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

AWS Exemples de SDK pour Java 1.x

Cette section contient un exemple de code pour les applications DAX utilisant le kit SDK pour Java 1.x.

Rubriques

- Utilisation de DAX avec le kit SDK AWS pour Java 1.x
- Modification d'une application du kit SDK pour Java 1.x pour utiliser DAX
- Interrogation d'index secondaires globaux avec le kit SDK pour Java 1.x

Utilisation de DAX avec le kit SDK AWS pour Java 1.x

Pour exécuter l'exemple Java pour Amazon DynamoDB Accelerator (DAX) sur votre instance Amazon EC2, procédez comme suit.

> **Note**
>
> Ces instructions s'appliquent aux applications utilisant le kit SDK AWS pour Java 1.x. Pour les applications utilisant le kit SDK AWS pour Java 2.x, consultez Java et DAX.

Pour exécuter l'exemple Java pour DAX

1. Installez le kit de développement Java (JDK).

   ```
   sudo yum install -y java-devel
   ```

2. Téléchargez le AWS SDK pour Java (fichier `.zip`) et extrayez-le.

   ```
   wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

   unzip aws-java-sdk.zip
   ```

3. Téléchargez la version la plus récente du client Java DAX (fichier `.jar`).

   ```
   wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
   latest.jar
   ```

   > **Note**
   >
   > Le client pour le kit SDK pour Java DAX est disponible sur Apache Maven. Pour plus d'informations, consultez Utilisation du client en tant que dépendance Apache Maven.

4. Définissez votre variable CLASSPATH. Dans l'exemple, remplacez *sdkVersion* par le numéro de version réel d'AWS SDK pour Java, par exemple, 1.11.112.

```
export SDKVERSION=sdkVersion

export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. Téléchargez le code source de l'exemple de programme (fichier .zip).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

   Une fois le téléchargement terminé, extrayez les fichiers source.

```
unzip TryDax.zip
```

6. Accédez au répertoire de code Java et compilez le code comme suit.

```
cd TryDax/java/
javac TryDax*.java
```

7. Exécutez le programme.

```
java TryDax
```

   Vous devez visualiser des résultats similaires à ce qui suit.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
Writing 10 items for partition key: 4
Writing 10 items for partition key: 5
Writing 10 items for partition key: 6
Writing 10 items for partition key: 7
```

```
Writing 10 items for partition key: 8
Writing 10 items for partition key: 9
Writing 10 items for partition key: 10

Running GetItem, Scan, and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

8. A l'étape précédente, vous avez exécuté le programme par rapport au point de terminaison DynamoDB. À présent, réexécutez le programme, mais cette fois, les opérations `GetItem`, `Query` et `Scan` sont traitées par votre cluster DAX.

   Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

   - Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Utilisation de l'AWS CLI – Entrez la commande suivante.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

À présent, réexécutez le programme, mais cette fois, spécifiez le point de terminaison du cluster en tant que paramètre de ligne de commande.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observez le reste de la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX qu'avec DynamoDB.

Pour plus d'informations sur ce programme, consultez les sections suivantes :

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

Utilisation du client en tant que dépendance Apache Maven

Suivez la procédure pour utiliser le client pour le kit SDK DAX pour Java dans votre application comme dépendance.

Pour utiliser le client comme dépendance Maven

1. Téléchargez et installez Apache Maven. Pour plus d'informations, consultez Downloading Apache Maven et Installing Apache Maven.
2. Ajoutez la dépendance de client Maven au fichier POM (Project Object Model) de votre application. Dans l'exemple, remplacez `x.x.x.x` par le numéro de version réel du client, par exemple : `1.0.200704.0`.

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

Le fichier `TryDax.java` contient la méthode `main`. Si vous exécutez le programme sans paramètre de ligne de commande, il crée un client Amazon DynamoDB et l'utilise pour toutes les opérations d'API. Si vous spécifiez un point de terminaison de cluster DynamoDB Accelerator (DAX) sur la ligne de commande, le programme crée également un client DAX et l'utilise pour les opérations `GetItem`, `Query` et `Scan`.

Vous pouvez modifier le programme de plusieurs façons :

- Utilisez le client DAX plutôt que le client DynamoDB. Pour plus d'informations, consultez Java et DAX.
- Attribuez un nom différent à la table de test.
- Modifiez le nombre d'éléments écrits en modifiant les paramètres `helper.writeData`. Le deuxième paramètre correspond au nombre de clés de partition et le troisième au nombre de clés de tri. Par défaut, le programme utilise 1–10 pour les valeurs de clé de partition, et 1–10 pour les valeurs de clé de tri, pour un total de 100 éléments écrits dans la table. Pour plus d'informations, consultez TryDaxHelper.java.
- Modifiez le nombre de tests `GetItem`, `Query` et `Scan`, puis modifiez leurs paramètres.

- Mettez en commentaire les lignes contenant `helper.createTable` et `helper.deleteTable` (si vous ne voulez pas créer et supprimer la table chaque fois que vous exécutez le programme).

> **Note**
>
> Pour exécuter ce programme, vous pouvez configurer Maven afin d'utiliser le kit SDK DAX pour Java et l'AWS SDK pour Java en tant que dépendances. Pour plus d'informations, consultez Utilisation du client en tant que dépendance Apache Maven.
> Sinon, vous pouvez télécharger et inclure le client Java DAX et l'AWS SDK pour Java dans votre chemin de classe. Consultez Java et DAX pour obtenir un exemple de configuration de votre variable `CLASSPATH`.

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }
```

```
        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

Le fichier `TryDaxHelper.java` contient des méthodes d'utilitaire.

Les méthodes `getDynamoDBClient` et `getDaxClient` fournissent des clients Amazon DynamoDB et DynamoDB Accelerator (DAX). Pour les opérations de plan de contrôle (`CreateTable`, `DeleteTable`) et les opérations d'écriture, le programme utilise le client DynamoDB. Si vous spécifiez un point de terminaison de cluster DAX, le programme principal crée un client DAX pour effectuer les opérations de lecture (`GetItem`, `Query`, `Scan`).

Les autres méthodes `TryDaxHelper` (`createTable`, `writeData`, `deleteTable`) sont destinées à la configuration et à la destruction de la table DynamoDB et de ses données.

Vous pouvez modifier le programme de plusieurs façons :

- Utilisez d'autres paramètres de débit alloué pour la table.
- Modifiez la taille de chaque élément écrit (voir la variable `stringSize` de la méthode `writeData`).
- Modifiez le nombre de tests `GetItem`, `Query` et `Scan`, ainsi que leurs paramètres.
- Mettez en commentaire les lignes contenant `helper.CreateTable` et `helper.DeleteTable` (si vous ne voulez pas créer et supprimer la table chaque fois que vous exécutez le programme).

> **Note**
>
> Pour exécuter ce programme, vous pouvez configurer Maven afin d'utiliser le kit SDK DAX pour Java et l'AWS SDK pour Java en tant que dépendances. Pour plus d'informations, consultez Utilisation du client en tant que dépendance Apache Maven.
> Sinon, vous pouvez télécharger et inclure le client Java DAX et l'AWS SDK pour Java dans votre chemin de classe. Consultez Java et DAX pour obtenir un exemple de configuration de votre variable `CLASSPATH`.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
```

```
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
```

```
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

Le fichier `TryDaxTests.java` contient des méthodes qui effectuent des opérations de lecture sur une table de test dans Amazon DynamoDB. Ces méthodes n'étant pas concernées par le mode d'accès aux données (à l'aide du client DynamoDB, ou du client DAX), il n'est pas nécessaire de modifier la logique d'application.

Vous pouvez modifier le programme de plusieurs façons :

- Modifiez la méthode `queryTest` afin qu'elle utilise une autre valeur pour `KeyConditionExpression`.
- Ajoutez un paramètre `ScanFilter` à la méthode `scanTest` de sorte que seule une partie des éléments vous soient retournés.

> **Note**
>
> Pour exécuter ce programme, vous pouvez configurer Maven afin d'utiliser le kit SDK DAX pour Java et l'AWS SDK pour Java en tant que dépendances. Pour plus d'informations, consultez Utilisation du client en tant que dépendance Apache Maven.
> Sinon, vous pouvez télécharger et inclure le client Java DAX et l'AWS SDK pour Java dans votre chemin de classe. Consultez Java et DAX pour obtenir un exemple de configuration de votre variable `CLASSPATH`.

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {

    void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations)
 {
        long startTime, endTime;
        System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" +
 sk);
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (Integer ipk = 1; ipk <= pk; ipk++) {
                    for (Integer isk = 1; isk <= sk; isk++) {
                        table.getItem("pk", ipk, "sk", isk);
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
```

```
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key " + pk + " and sort keys between
" + sk1 + " and " + sk2);
        Table table = client.getTable(tableName);

        HashMap<String, Object> valueMap = new HashMap<String, Object>();
        valueMap.put(":pkval", pk);
        valueMap.put(":skval1", sk1);
        valueMap.put(":skval2", sk2);

        QuerySpec spec = new QuerySpec()
                .withKeyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                .withValueMap(valueMap);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<QueryOutcome> items = table.query(spec);

            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);
```

```
        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {

                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

Modification d'une application du kit SDK pour Java 1.x pour utiliser DAX

Si vous disposez déjà d'une application Java utilisant Amazon DynamoDB, vous devez la modifier pour qu'elle puisse accéder à votre cluster DynamoDB Accelerator (DAX). Le client Java étant similaire au client de bas niveau DynamoDB inclus dans l'AWS SDK pour Java, vous n'avez pas besoin de réécrire entièrement l'application.

> **Note**
>
> Ces instructions s'appliquent aux applications utilisant le kit SDK AWS pour Java 1.x. Pour les applications utilisant le kit SDK AWS pour Java 2.x, consultez Modification d'une application existante pour utiliser DAX .

Supposez que vous disposez d'une table DynamoDB nommée `Music`. La clé de partition de la table est `Artist` et sa clé de tri est `SongTitle`. Le programme suivant lit un élément directement dans la table `Music`.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);

        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
    }
}
```

Pour modifier le programme, vous remplacez le client DynamoDB par un client DAX.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
```

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

Utilisation de l'API de document DynamoDB

L'AWS SDK pour Java fournit une interface de document pour DynamoDB. L'API Document agit en tant qu'encapsuleur englobant le client DynamoDB de bas niveau. Pour plus d'informations, consultez Interfaces de document.

L'interface de document peut également être utilisée avec le client DAX de bas niveau comme dans l'exemple suivant.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {
```

```
    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

Client asynchrone DAX

`AmazonDaxClient` est synchrone. Pour une opération d'API DAX de longue durée, par exemple une opération `Scan` exécutée sur une table volumineuse, ce client peut bloquer l'exécution du programme jusqu'à la fin de l'opération. Si votre programme doit effectuer d'autres tâches au cours d'une opération d'API DAX, vous pouvez utiliser `ClusterDaxAsyncClient` à la place.

Le programme suivant montre comment utiliser `ClusterDaxAsyncClient` avec `Future` de Java pour implémenter une solution sans blocage.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import com.amazon.dax.client.dynamodbv2.ClientConfig;
```

```
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
   Thread.sleep(3000);
  }
  // The results should be ready by now

  try {
   call.get();

  } catch (ExecutionException ee) {
   // Futures always wrap errors as an ExecutionException.
   // The *real* exception is stored as the cause of the
   // ExecutionException
   Throwable exception = ee.getCause();
   System.out.println("Error getting item: " + exception.getMessage());
  }
```

```
  // Async callbacks
  call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
 {

   @Override
   public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
    System.out.println("Result: " + getItemResult);
   }

   @Override
   public void onError(Exception e) {
    System.out.println("Unable to read item");
    System.err.println(e.getMessage());
    // Callers can also test if exception is an instance of
    // AmazonServiceException or AmazonClientException and cast
    // it to get additional information
   }

  });
  call.get();

 }
}
```

Interrogation d'index secondaires globaux avec le kit SDK pour Java 1.x

Vous pouvez utiliser Amazon DynamoDB Accelerator (DAX) pour interroger des index secondaires globaux à l'aide d'interfaces de programmation DynamoDB.

L'exemple suivant montre comment utiliser DAX pour interroger l'index secondaire global `CreateDateIndex` créé dans Exemple : index secondaires globaux utilisant l'API de document AWS SDK pour Java.

La classe `DAXClient` instancie les objets clients requis pour interagir avec les interfaces de programmation DynamoDB.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import software.amazon.dynamodb.AmazonDynamoDB;
```

```
public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

Vous pouvez interroger un index secondaire global selon les méthodes suivantes :

- Utilisez la méthode `queryIndex` sur la classe `QueryIndexDax` définie dans l'exemple suivant. `QueryIndexDax` utilise en tant que paramètre l'objet client renvoyé par la méthode `getDaxDocClient` sur la classe `DaxClient`.
- Si vous utilisez l'interface de persistance des objets, utilisez la méthode `queryIndexMapper` sur la classe `QueryIndexDax` définie dans l'exemple suivant. `queryIndexMapper` utilise en tant que paramètre l'objet client renvoyé par la méthode `getDaxMapperClient` définie sur la classe `DaxClient`.

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
```

```
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n*******************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
   querySpec.withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withValueMap(new ValueMap().withString(":v_date",
 "2013-11-01").withString(":v_issue", "A-"));
   items = index.query(querySpec);
  } else {
   System.out.println("\nNo valid index name provided");
   return;
  }

  Iterator<Item> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   System.out.println(iterator.next().toJSONPretty());
  }
```

```
 }

 //This is used to query Index using the high-level mapper interface.
 public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
 indexName) {
  HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
  eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
  eav.put(":v_issue", new AttributeValue().withS("A-"));
  DynamoDBQueryExpression<CreateDate> queryExpression = new
 DynamoDBQueryExpression<CreateDate>()
     .withIndexName("CreateDateIndex").withConsistentRead(false)
     .withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withExpressionAttributeValues(eav);

  List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
  Iterator<CreateDate> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   CreateDate iterObj = iterator.next();
   System.out.println(iterObj.getCreateDate());
   System.out.println(iterObj.getIssueId());
  }
 }
}
```

La définition de classe ci-dessous représente la table Issues et est utilisée dans la méthode `queryIndexMapper`.

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;
```

```
 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

AWS Exemples de SDK pour Go 1.x

Cette section contient un exemple de code pour les applications DAX utilisant Go 1.x.

Rubriques

- Kit SDK DAX pour Go

Kit SDK DAX pour Go

Suivez cette procédure pour exécuter l'exemple d'application SDK Amazon DynamoDB Accelerator (DAX) pour Go sur votre instance Amazon. EC2

Pour exécuter l’exemple du kit SDK pour Go pour DAX

1. Configurez le SDK pour Go sur votre instance EC2 Amazon :

   a. Installez le langage de programmation Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

b. Testez que Golang est installé et fonctionne correctement.

```
go version
```

Un message comme celui-ci devrait s'afficher.

```
go version go1.15.5 linux/amd64
```

Les instructions restantes reposent sur la prise en charge du module qui est devenu le module par défaut avec Go version 1.13.

2. Installez l'exemple d'application Golang.

```
go get github.com/aws-samples/aws-dax-go-sample
```

3. Exécutez les programmes Golang suivants. Le premier programme crée une table DynamoDB nommée `TryDaxGoTable`. Le deuxième programme écrit des données dans la table.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command put-item
```

4. Exécutez les programmes Golang suivants.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

5. A l'étape précédente, vous avez exécuté les programmes par rapport au point de terminaison DynamoDB. A présent, réexécutez-les mais, cette fois, les opérations `GetItem`, `Query` et `Scan` sont traitées par votre cluster DAX.

   Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes :

   - Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - À l'aide de AWS CLI— Entrez la commande suivante.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   À présent, réexécutez les programmes, mais cette fois, spécifiez le point de terminaison du cluster en tant que paramètre de ligne de commande.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

Observez le reste de la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX qu'avec DynamoDB.

6. Exécutez le programme Golang suivant pour supprimer `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

AWS Exemples de SDK pour Node.js 2.x

Cette section contient un exemple de code pour les applications DAX utilisant le AWS SDK pour Node.js 2.x.

Rubriques

- Node.js et DAX

Node.js et DAX

Suivez ces étapes pour exécuter l'exemple d'application Node.js sur votre EC2 instance Amazon.

Pour exécuter l'exemple Node.js pour DAX

1. Configurez Node.js sur votre EC2 instance Amazon, comme suit :

   a. Installez le gestionnaire de version de nœud (`nvm`).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Utilisez nvm pour installer Node.js.

   ```
   nvm install 12.16.3
   ```

   c. Testez que Node.js est installé et fonctionne correctement.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Le message suivant doit s'afficher.

```
Running Node.js v12.16.3
```

2. installez le client Node.js DAX à l'aide du gestionnaire de package de nœud (`npm`).

```
npm install amazon-dax-client
```

3. Téléchargez le code source de l'exemple de programme (fichier `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Une fois le téléchargement terminé, extrayez les fichiers source.

```
unzip TryDax.zip
```

4. Exécutez les programmes Node.js suivants : Le premier programme crée une table Amazon DynamoDB nommée `TryDaxTable`. Le deuxième programme écrit des données dans la table.

```
node 01-create-table.js
node 02-write-data.js
```

5. Exécutez les programmes Node.js suivants :

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Notez les informations de durée, soit le nombre de millisecondes requis pour les tests `GetItem`, `Query` et `Scan`.

6. A l'étape précédente, vous avez exécuté les programmes par rapport au point de terminaison DynamoDB. Réexécutez-les mais, cette fois, les opérations `GetItem`, `Query` et `Scan` sont traitées par votre cluster DAX.

Pour déterminer le point de terminaison de votre cluster DAX, choisissez l'une des options suivantes.

- Utilisation de la console DynamoDB – Choisissez votre cluster DAX. Le point de terminaison du cluster s'affiche dans la console, comme dans l'exemple suivant.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- À l'aide de la commande AWS CLI—Entrez la commande suivante.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Le point de terminaison du cluster apparaît dans la sortie, comme dans l'exemple suivant.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

À présent, réexécutez les programmes, mais cette fois, spécifiez le point de terminaison du cluster en tant que paramètre de ligne de commande.

```
node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observez le reste de la sortie et notez les informations de durée. Les délais écoulés pour `GetItem`, `Query` et `Scan` devraient être sensiblement inférieurs avec DAX qu'avec DynamoDB.

7. Exécutez le programme Node.js suivant pour supprimer `TryDaxTable`.

```
node 06-delete-table
```

Pour plus d'informations sur ces programmes, consultez les sections suivantes :

- 01-create-table.js
- 02-write-data.js
- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js

- 06-delete-table.js

01-create-table.js

Le programme `01-create-table.js` crée une table (`TryDaxTable`). Les programmes Node.js restants de cette section dépendent de cette table.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
```

```
    console.log(
      "Created table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

02-write-data.js

Le programme `02-write-data.js` écrit des données de test dans `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();

var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
      },
    };

    //
    //put item
```

```
    ddbClient.put(params, function (err, data) {
      if (err) {
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

03-getitem-test.js

Le programme `03-getitem-test.js` exécute des opérations `GetItem` sur `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
```

```
  var startTime = new Date().getTime();

  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

04-query-test.js

Le programme `04-query-test.js` exécute des opérations `Query` sur `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");
```

```
var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
```

```
    } else {
      // Query succeeded
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

05-scan-test.js

Le programme `05-scan-test.js` exécute des opérations `Scan` sur `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";
```

```
var iterations = 5;

var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
  (endTime - startTime) / iterations,
  "ms"
);
```

06-delete-table.js

Le programme `06-delete-table.js` supprime `TryDaxTable`. Exécutez ce programme après avoir fini les tests.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});
```

```
var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

Historique du document pour DynamoDB

Le tableau ci-après décrit les modifications importantes apportées à chaque édition du Guide du développeur DynamoDB à partir du 3 juillet 2018. Pour recevoir une notification sur les mises à jour apportées à cette documentation, vous pouvez vous abonner au flux RSS (dans le coin supérieur gauche de cette page).

Modification	Description	Date
DAX prend désormais AWS PrivateLink en charge les régions commerciales et chinoises	Ajout de conseils d'utilisation AWS PrivateLink avec DynamoDB Accelerator (DAX) afin d'aider les utilisateurs à établir une connectivité privée et sécurisée avec les points de terminaison du plan de contrôle DAX au sein de celui-ci. Région AWS Pour plus d'informations, consultez Générer du code d'infrastructure pour Amazon Console-to-Code DynamoDB à l'aide de.	30 octobre 2025
DynamoDB prend désormais en charge la version 6 du protocole Internet () IPv6	DynamoDB prend désormais en charge les adresses du protocole Internet version 6 IPv6 () dans votre Amazon VPC lorsque vous vous connectez aux tables, aux flux et au DAX DynamoDB, y compris avec les points de terminaison de passerelle et d'interface. AWS PrivateLink Vous pouvez l'utiliser IPv6 pour simplifier votre infrastru	3 octobre 2025

	cture réseau et répondre aux exigences de conformité grâce à un réseau qui prend en charge IPv4 les deux IPv6.	
Documentation ajoutée pour la nouvelle Console-to-Code fonctionnalité de DynamoDB	Console-to-Code transform e les étapes manuelles de création de tables DynamoDB en AWS Management Console code d'infrastructure automatis é et reproductible dans plusieurs formats tels que CDK et. AWS CloudForm ation Pour plus d'informations, consultez Générer du code d'infrastructure pour Amazon Console-to-Code DynamoDB à l'aide de.	23 septembre 2025
Lancement de DynamoDB Local version 3.1.0	Cette mise à jour améliore les performances des requêtes PartiQL, y notamment la dépendance au temps de Joda.	14 septembre 2025

DynamoDB a amélioré ses fonctionnalités de diagnostic de régulation grâce à un CloudWatch nouveau mode Contributor Insights, à l'ajout de détails sur les exceptions de régulation et à des mesures ciblées CloudWatch	DynamoDB fournit des diagnostics de régulation améliorés avec un `Throttled keys` nouveau mode exclusif CloudWatch dans Contributor Insights qui suit spécifiquement les clés de partition les plus limitées, des exceptions de régulation plus détaillées qui incluent des champs `ThrottlingReason` structurés, fournissant des codes de raison et des ressources spécifiques pour aider à identifier exactement quelle ARNs ressource est limitée et pourquoi, et 16 nouvelles mesures CloudWatch ciblées correspondant à différents scénarios de régulation, pour un suivi précis d'événements de régulation spécifiques. Pour plus d'informations, consultez Troubleshooting throttling in DynamoDB.	14 août 2025

Limite accrue pour passer d'un mode de capacité à un autre	La limite pour le passage d'une table du mode capacité provisionnée au mode capacité à la demande a été augmentée, passant d'une fois toutes les 24 heures à quatre fois toutes les 24 heures. Cette modification offre une plus grande flexibilité pour les charges de travail qui nécessitent le chargement de gros volumes de données plusieurs fois par jour ou qui doivent faire face à des augmentations de trafic inattendues. Pour plus d'informations, consultez Considérations relatives au changement de mode de capacité dans DynamoDB.	12 août 2025
Publication de la version 3.0 de DynamoDB Local, qui migre du SDK AWS pour Java V1 vers le V2	Cette mise à jour supprime la dépendance V1, améliore les performances et s'aligne sur la structure du package AWS SDK V2.	17 juillet 2025

Ajout de la prise en charge de l'optimisation de la découverte de partitions dans DynamoDB Streams	Le nouveau mécanisme de découverte de partitions réduit le temps de découverte initial, améliore les performances de traitement à un état stable, fournit une visibilité sur l'état du traitement et maintient une latence p100 constante pour les opérations à grande échelle. Pour plus d'informations, consultez AWS PrivateLink for Amazon DynamoDB Streams.	17 juillet 2025
Ajout de la prise en charge MRSC (forte cohérence multirégionale) pour les tables globales DynamoDB	DynamoDB prend désormais en charge MRSC (forte cohérence multirégionale) pour les tables globales avec région témoin en option. MRSC permet les lectures fortement cohérentes dans les régions avec un objectif de point de reprise (RPO) égal à zéro durant des défaillances régionales. Les tables globales MRSC doivent être déployées dans exactement trois régions, avec trois réplicas ou deux réplicas et un témoin. Pour plus d'informations et pour connaître les régions disponibles, voir Forte cohérence multirégionale (MRSC).	30 juin 2025

Adaptateur Amazon DynamoDB Streams Kinesis pour Client Library (KCL) 2.x	DynamoDB prend désormais en charge un adaptateur DynamoDB Streams mis à niveau compatible avec KCL 2.x et SDK v2. AWS Pour plus d'informations, consultez la section Migration de KCL1 .x vers KCL 3.x.	11 juin 2025
Ajout de la prise en charge de l'exécution locale de DynamoDB dans AWS CloudShell	Vous pouvez désormais exécuter DynamoDB local directement dans votre navigateur sans télécharger ni installer le logiciel AWS Management Console . Pour plus d'informations, consultez Deploying DynamoDB locally on your computer.	19 mai 2025
Ajout de la prise en charge des types d'instances r7i dans DAX	Options de type d'instance étendues pour DAX afin d'offrir plus de flexibilité en termes de performances et de capacité. Pour plus de détails sur les nouveaux types d'instances r7i et leurs fonctionnalités, consultez Chiffrement au repos DAX.	29 avril 2025

La nouvelle stratégie d'accès `AmazonDynamoDBFull Access_v2` délimitée remplace `AmazonDyn amoDBFullAccess` .	À compter du 25 avril 2025, DynamoDB a rendu cette stratégie `AmazonDyn amoDBFullAccess` obsolète et l'a remplacée par une stratégie délimitée nommée `AmazonDyn amoDBFullAccess_v2` . Pour plus d'informations, consultez la politique AWS gérée : AmazonDynamo DBFull Access_v2.	28 avril 2025
DynamoDB AWS PrivateLink introduit le support pour DynamoDB Streams	AWS AWS PrivateLink prise en charge ajoutée de DynamoDB Streams, permettant aux clients d'invoquer DynamoDB Streams APIs depuis Amazon Virtual Private Cloud (VPC) via une connectivité réseau privée. Cette fonctionnalité permet un accès simplifié au réseau privé sans passer par l'Internet public, ce qui répond aux exigences de conformité et de sécurité pour les charges de travail DynamoDB. Pour plus d'informations, consultez AWS PrivateLink for Amazon DynamoDB Streams.	26 mars 2025

Le kit SDK DAX pour JS 3.x et Go 2.x est désormais disponible	Le SDK DynamoDB Accelerator (DAX) pour JS 3.x est désormais disponible et est compatible avec le SDK pour Java 3.x. AWS	17 mars 2025
Mode de capacité à la demande NoSQL Workbench 3.13.5 pour la mise en place de paramètres de table par défaut	Lorsque vous créez une table avec les paramètres par défaut, DynamoDB crée une table qui utilise le mode de capacité à la demande au lieu du mode de capacité provisionnée.	24 février 2025
Nouvelles bonnes pratiques pour la configuration de votre client DAX	Publication d'une nouvelle rubrique sur les meilleures pratiques du guide prescriptif DAX pour la configuration de votre client DAX. Pour plus d'informations, consultez Configuration de votre client DAX.	17 février 2025
Nouvelles bonnes pratiques pour les opérations de traitement des données en bloc	Publication de nouvelles rubriques sur les meilleures pratiques relatives à l'utilisation de modèles de données complexes dans DynamoDB. Pour plus d'informations, consultez Bonnes pratiques liées à l'utilisation d'opérations de données groupées dans DynamoDB.	9 janvier 2025

Amazon DynamoDB prend désormais en charge la restauration point-in-time configurable (PITR)	DynamoDB prend désormais en charge les périodes de reprise configurables pour PITR. Vous pouvez désormais définir une période PITR comprise entre 1 et 35 jours pour chaque table. Pour plus d'informations, consultez la section Point-in-time sauvegardes pour DynamoDB.	7 janvier 2025
Publication d'une nouvelle rubrique sur l'intégration d'Amazon Managed Streaming for Apache Kafka à Amazon DynamoDB	Découvrez comment Amazon Managed Streaming for Apache Kafka s'intègre à Amazon DynamoDB en lisant les données des rubriques Apache Kafka et en les stockant dans DynamoDB. Pour plus d'informations, consultez Intégration de DynamoDB à Amazon Managed Streaming for Apache Kafka.	26 décembre 2024
DynamoDB introduit la prise en charge d'une intégration zéro ETL avec AI Lakehouse SageMaker	DynamoDB introduit une intégration zéro ETL qui automatise l'extraction et le chargement des données depuis Amazon DynamoDB dans le lac de données d'un client. Pour plus d'informations, consultez la section Intégration de DynamoDB Zero-ETL à AI Lakehouse. SageMaker	3 décembre 2024

Les tables globales DynamoDB prennent désormais en charge la forte cohérence multirégionales	Grâce à une forte cohérence multirégionale, vous pouvez créer des applications multirégionales à haute disponibilité avec un objectif de point de reprise (RPO) nul, pour atteindre le plus haut niveau de résilience. Pour plus d'informations, consultez Global tables with multi-Region strong consistency.	3 décembre 2024
Mise à jour de la politique gérée par DynamoDB	Ajout de deux nouvelles autorisations à la politique gérée par `AmazonDynamoDBReadOnlyAccess` : `dynamodb:GetAbacStatus` et `dynamodb:UpdateAbacStatus`. Ces autorisations vous permettent de consulter le statut ABAC et d'activer ABAC pour vous Compte AWS dans la région actuelle. Pour plus d'informations, consultez la section Politique AWS gérée : AmazonDynamo DBReadOnlyAccess.	18 novembre 2024

DynamoDB introduit la prise en charge pour le Contrôle d’accès par attributs (ABAC)	ABAC est une stratégie d'autorisation qui vous permet de définir des autorisations d'accès en fonction des balises associées aux utilisateurs, aux rôles et aux AWS ressources. ABAC utilise des conditions basées sur les balises dans vos politiques Gestion des identités et des accès AWS (IAM) ou dans d'autres politiques pour autoriser ou refuser des actions spécifiques sur vos tables ou index lorsque les balises des principaux IAM correspondent aux balises des tables. Pour plus d’informations, consultez Utilisation du contrôle d’accès par attributs avec DynamoDB.	18 novembre 2024
DynamoDB introduit un débit optimal pour les tables à la demande et provisionnées	DynamoDB prend désormais en charge le débit chaud. Le débit chaud fournit une visibilité sur le nombre d’opérations de lecture et d’écriture que votre table DynamoDB peut instantanément prendre en charge, ainsi que la possibilité de préchauffer vos tables DynamoDB. Pour plus d’informations, consultez DynamoDB warm throughput.	13 novembre 2024

Possibilité d'utiliser des enregistrements Amazon DynamoDB Streams avec Apache Flink	Vous pouvez désormais utiliser les enregistrements Amazon DynamoDB Streams avec Apache Flink et tirer parti du service géré Amazon pour Apache Flink pour créer et gérer rapidement des applications de traitement de flux. end-to-end Pour plus d'informations, consultez DynamoDB Streams et Apache Flink.	12 novembre 2024
Publication de nouvelles rubriques de facturation pour les tables globales et les sauvegardes	Publication de nouvelles rubriques concernant la facturation des tables globales et des sauvegardes. Pour plus d'informations, consultez Présentation de la facturation Amazon DynamoDB pour les tables globales et Présentation de la facturation des sauvegardes pour Amazon DynamoDB.	16 octobre 2024
Intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift	L'intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift fournit un pipeline ETL entièrement géré sans programmation avec réplication de DynamoDB vers Amazon Redshift. Pour plus d'informations, consultez Intégration zéro ETL d'Amazon DynamoDB à Amazon Redshift.	15 octobre 2024

Mise à jour uniquement de la documentation pour l'ajout d'une rubrique sur l'utilisation de l'IA générative avec DynamoDB	Publication d'une nouvelle rubrique qui fournit des informations sur l'utilisation de l'IA générative avec DynamoDB, notamment des exemples de cas d'utilisation de l'IA générative dans DynamoDB. Pour plus d'informations, consultez Utilisation de l'IA générative avec DynamoDB.	11 octobre 2024
Le SDK prend désormais en charge les points de terminaison basés sur des AWS comptes	Ajout de documentation pour les points de terminaison basés sur les comptes et les paramètres `ACCOUNT_ID_ENDPOINT_MODE` pour les clients du kit SDK. Pour plus d'informations, consultez la section Support du SDK pour les points de terminaison AWS basés sur un compte.	3 septembre 2024
Nouvelle expérience de mise en route	L'expérience de mise en route a été repensée pour consolider les informations et vous permettre de démarrer avec une intégration plus rapide. Pour plus d'informations, consultez Mise en route avec DynamoDB.	1er août 2024
Expansion DAX vers de nouvelles régions en Espagne et en Suède	DAX est désormais disponible dans les régions d'Espagne et de Suède. Pour plus d'informations, consultez Composants de cluster DAX.	30 juillet 2024

Restructuration et consolidation de la documentation de sauvegarde et de restauration dans DynamoDB	Le Guide du développeur DynamoDB propose une nouvelle structure de sauvegarde et de restauration. Pour plus d'informations, consultez Sauvegarde et restauration pour DynamoDB.	2 juillet 2024
Réécriture de la rubrique Qu'est-ce qu'Amazon DynamoDB ?	Publication d'une version révisée et mise à jour de la rubrique Qu'est-ce qu'Amazon DynamoDB ?. Pour plus d'informations, consultez Qu'est-ce qu'Amazon DynamoDB ?.	21 juin 2024
Intégrez DynamoDB Streams avec EventBridge	Publication d'une nouvelle rubrique sur l'intégration de DynamoDB Streams avec. EventBridge Pour plus d'informations, consultez la section Intégration avec EventBridge.	21 juin 2024
Recommandations prescriptives concernant DAX	Publication d'une nouvelle rubrique sur les meilleures pratiques qui fournit des informations complètes sur l'utilisation efficace de DynamoDB Accelerator. Cette rubrique couvre l'optimisation des performances, la gestion des coûts et les meilleures pratiques opérationnelles. Pour plus d'informations, consultez Recommandations prescriptives concernant DAX.	3 juin 2024

Migration d’une table DynamoDB d’un compte à un autre	Ajout d’une nouvelle rubrique sur la migration des tables DynamoDB d’un compte à un autre. Pour plus d’informations, consultez Migration d’une table DynamoDB d’un compte à un autre.	29 mai 2024
Restructuration et consolidation de la documentation relative à la surveillance et à la journalisation de DynamoDB	La nouvelle structure de surveillance et de journalisation dans DynamoDB comprend trois chapitres concis consacrés aux métriques, aux opérations de journalisation et aux informations des contributeurs.	3 mai 2024

Restructuration et consolidation de la documentation relative au mode de capacité DynamoDB	Le guide DynamoDB inclut désormais un nouveau chapitre qui contient toutes les informations relatives aux modes de capacité DynamoDB : à la demande and provisionné. Avec cette mise à jour, la rubrique Considérations à prendre en compte lors du changement de mode read/write capacité a été déplacée dans le chapitre des meilleures pratiques. Cette rubrique est désormais renommée Considérations relatives au changemen t de mode de capacité et elle inclut des informations détaillées sur les meilleures pratiques lors du passage d'un mode de capacité à un autre. En outre, le guide contient désormais un nouveau chapitre qui inclut toutes les informations relatives aux opérations de lecture et d'écriture DynamoDB, ainsi que les unités de capacité utilisées pour les opération s de lecture et d'écriture. Pour plus d'informations, consultez Capacité de débit DynamoDB, Considérations relatives au changement de mode de capacité et Lectures et écritures DynamoDB.	1 mai 2024

Nombre maximal de demandes à la demande	Vous pouvez désormais spécifier le nombre maximum de demandes à la demande qu'une table, un index ou les deux peuvent effectuer. La spécification du débit maximal à la demande vous aidera à limiter votre utilisation et vos coûts au niveau de la table et à vous protéger contre une augmentation involontaire des ressources utilisées. Pour plus d'informations, consultez Débit maximal DynamoDB pour les tables à la demande.	1 mai 2024
Améliorations du créateur d'opérations NoSQL Workbench	NoSQL Workbench inclut désormais la prise en charge native du mode sombre. Amélioration des opérations sur les tables et les éléments dans le créateur d'opérations. Les informations sur les résultats des éléments et les demandes du créateur d'opérations sont disponibles au format JSON. Pour plus d'informations, consultez Créateur d'opérations NoSQL Workbench.	24 avril 2024

Politiques basées sur les ressources pour les ressources Amazon DynamoDB	DynamoDB prend en charge les politiques basées sur les ressources pour les tables, les index et les flux. Les politiques basées sur les ressources vous permettent de définir les autorisations d'accès en spécifiant qui a accès à chaque ressource et les actions que cette personne est autorisée à effectuer sur chaque ressource. Pour plus d'informations, consultez Utilisation de politiques basées sur des ressources pour DynamoDB.	20 mars 2024
Mise à jour de la politique gérée par DynamoDB	Ajout d'une nouvelle autorisation `dynamodb:GetResourcePolicy` à la politique gérée par `AmazonDynamoDBReadOnlyAccess`. Cette autorisation permet d'accéder aux politiques basées sur les ressources associées aux ressources DynamoDB. Pour plus d'informations, consultez la section Politique AWS gérée : AmazonDynamo DBRead OnlyAccess.	20 mars 2024

AWS PrivateLink pour Amazon DynamoDB	Amazon DynamoDB prend désormais en charge. AWS PrivateLink Vous pouvez ainsi simplifier la connectivité réseau privé entre les clouds privés virtuels (VPCs), DynamoDB et vos centres de données locaux à l'aide de points de terminaison VPC d'interface et d'adresses IP privées. AWS PrivateLink Pour plus d'informations, consultez AWS PrivateLink pour DynamoDB.	19 mars 2024
Programmation avec JavaScript guide	Amazon DynamoDB présente un guide de programmation pour. AWS SDK pour JavaScript Découvrez AWS SDK pour JavaScript, les couches d'abstraction, la configuration de connexion, le traitement des erreurs, la définition de politiques de nouvelle tentative, la gestion keep-alive, et bien d'autres fonctionnalités. Pour plus d'informations, consultez la section Programmation avec JavaScript.	6 mars 2024

Programmation avec AWS SDK for Java 2.x guide	Création d'un nouveau guide de programmation qui décrit en détail les interfaces de haut niveau, de bas niveau et de document, les clients HTTP et leur configuration, la gestion des erreurs, et aborde les paramètres de configuration les plus courants à prendre en compte lors de l'utilisation du kit SDK for Java 2.x. Pour plus d'informations, consultez Programmer Amazon AWS SDK for Java 2.x DynamoDB avec.	5 mars 2024
Clonage de tables avec NoSQL Workbench	Permettez aux développeurs d'utiliser NoSQL Workbench pour copier ou cloner des tables entre des environnements de développement et des régions (DynamoDB Local et DynamoDB web). Pour plus d'informations, consultez Clonage de tables avec NoSQL Workbench.	26 février 2024

Guide de programmation avec Python	Création d'un nouveau guide qui décrit en détail les bibliothèques de haut niveau et de bas niveau et aborde les paramètres de configuration les plus courants à prendre en compte lors de l'utilisation du kit SDK Python. Pour plus d'informations, consultez Programmation avec Python.	5 janvier 2024
Réécriture de la rubriques Durée de vie (TTL)	Réécriture complète de la section TTL du guide. Le nouveau guide vous aide à démarrer avec le TTL en fournissant des extraits de ready-to-use code en cours de route. Les extraits de code actuellement fournis sont en Python et Javascript. Pour plus d'informations, consultez TTL.	20 décembre 2023
Meilleures pratiques pour comprendre vos rapports AWS de facturation et d'utilisation	Ajout d'une nouvelle section qui précise les différents types d'utilisation et les frais associés à ces types d'utilisation dans DynamoDB. Pour plus d'informations, consultez Billing and usage reports.	15 décembre 2023

Intégration d'Amazon DynamoDB Zero-ETL à Amazon Service OpenSearch	Amazon DynamoDB prend désormais en charge l'intégration zéro ETL avec OpenSearch Amazon Service, ce qui vous permet d'effectuer une recherche sur vos données DynamoDB en les répliquant et en les transformant automatiquement sans code ni infrastructure personnalisés. Pour plus d'informations, consultez la section Intégration de DynamoDB Zero-ETL à Amazon Service. OpenSearch	28 novembre 2023
Migration à partir d'une base de données relationnelle vers DynamoDB	Création d'un guide de migration pour aider les utilisateurs à migrer vers DynamoDB à partir d'une base de données relationnelle.	27 novembre 2023

Génération de données d’exemple avec NoSQL Workbench	NoSQL Workbench pour Amazon DynamoDB prend désormais en charge la création de modèles de données directement à partir de modèles de données d’exemple afin de vous aider à concevoir des schémas de données pour vos charges de travail. Vous pouvez utiliser cette fonctionnalité pour vous familiariser avec les bonnes pratiques de la modélisation de données NoSQL lors de la création d’applications sur DynamoDB.	28 septembre 2023
Exportation incrémentielle vers S3	Vous pouvez désormais exporter des données qui ont été insérées, mises à jour ou supprimées par petits incréments. Grâce à l'exportation incrémentielle, vous pouvez exporter des données modifiées allant de quelques mégaoctets à plusieurs téraoctets en quelques clics dans la console de AWS gestion, un appel d'API ou l'interface de ligne de commande. AWS	26 septembre 2023

Modélisation de données pour DynamoDB	Vous pouvez désormais en savoir plus sur la modélisation des données à l'aide d'exemples DynamoDB axés sur des cas d'utilisation spécifiques, leurs modèles d'accès step-by-step et des conseils pour réaliser ces modèles d'accès.	14 juillet 2023
Section Dépannage	Vous pouvez désormais accéder au contenu de résolution des problèmes liés à la latence et à la limitation susceptibles de survenir dans vos tables DynamoDB.	13 mars 2023
Protection contre la suppression pour Amazon DynamoDB	La protection contre la suppression est désormais disponible pour les tables Amazon DynamoDB dans toutes les régions AWS . DynamoDB vous permet désormais de protéger vos tables contre toute suppression accidentelle lors de l'exécution d'opérations de gestion de table régulières.	8 mars 2023

CloudFormation prise en charge du KDSD dans les tableaux globaux	Amazon Kinesis Data Streams for DynamoDB CloudFormation prend désormais en charge les tables globales DynamoDB, ce qui signifie que vous pouvez activer le streaming vers un Amazon Kinesis Data Streams sur vos tables globales DynamoDB avec des modèles. CloudFormation	15 février 2023
DynamoDB Local prend en charge 100 actions par transaction	Vous pouvez désormais effectuer jusqu'à 100 actions en une seule transaction sur DynamoDB Local.	9 février 2023
Utilisation du cadre DynamoDB Well-Architected pour optimiser votre charge de travail DynamoDB	Vous pouvez désormais utiliser le cadre Well-Architected de DynamoDB, un ensemble de principes de conception et de conseils pour concevoir des charges de travail DynamoDB bien architecturées.	3 février 2023
Disponibilité partiQL GovGloud	PartiQL : un langage de requête compatible avec SQL pour Amazon DynamoDB est désormais pris en charge dans (USA Est) et (USA Ouest). AWS GovCloud AWS GovCloud	21 décembre 2022

Suite d’installation unique pour NoSQL Workbench et DynamoDB Local	NoSQL Workbench for DynamoDB inclut désormais un processus d’installation guidé de DynamoDB Local guidé afin de rationaliser la configuration de votre environnement de développement DynamoDB Local.	6 décembre 2022
Importer des données en bloc à partir de S3	Amazon DynamoDB facilite désormais la migration et le chargement de données dans de nouvelles tables DynamoDB en prenant en charge les importations de données en bloc à partir d’Amazon S3.	18 août 2022
Intégration améliorée à Service Quotas	Service Quotas vous permet désormais de gérer de manière proactive les quotas de votre compte et de vos tables. Vous pouvez afficher les valeurs actuelles, définir des alarmes lorsque votre utilisation d’un quota dépasse un seuil configurable, etc.	15 juin 2022
NoSQL Workbench ajoute la prise en charge des tables et des index globaux	Vous pouvez désormais utiliser NoSQL Workbench pour les opérations de table et d'index secondaire global (GSI) telles CreateTable que, et. UpdateTable DeleteTable	2 juin 2022

La classe de tables à accès standard peu fréquent est désormais disponible en Chine	La classe de tables Amazon DynamoDB Standard-Inrequent Access est disponible dans les régions chinoises. Réduisez vos coûts DynamoDB jusqu'à 60 %, en utilisant cette nouvelle classe de tables pour les tables qui stockent des données rarement consultées.	18 avril 2022
Augmentation des quotas de service par défaut et des opérations de gestion des tables	DynamoDB a augmenté le quota par défaut pour le nombre de tables par compte et par région de 256 à 2 500 tables, et le nombre d'opérations de gestion de tables simultanées est passé de 50 à 500.	9 mars 2022
Limitation facultative des éléments avec PartiQL pour DynamoDB	DynamoDB peut limiter le nombre d'éléments traités dans PartiqL pour les opérations DynamoDB en tant que paramètre facultatif pour chaque demande.	8 mars 2022

AWS Backup intégration disponible dans les régions de Chine (Pékin et Ningxia)	AWS Backup s'intègre désormais à DynamoDB dans les régions de Chine (Beijing et Ningxia). Vous pouvez répondre plus facilement aux exigences de conformité et de continuité des activités grâce à des fonctionnalités de sauvegarde améliorées AWS Backup, telles que les sauvegardes entre comptes et entre régions.	26 janvier 2022
Informations sur la capacité de débit via les appels d'API PartiQL	DynamoDB peut renvoyer la capacité de débit consommée par les appels d'API PartiQL pour vous aider à optimiser vos requêtes et vos coûts de débit.	18 janvier 2022
AWS Backup intégration	DynamoDB vous aide désormais à répondre plus facilement aux exigences de conformité et de continuit é d'activité grâce à des fonctionnalités de sauvegard e améliorées dans AWS Backup, telles que les sauvegardes entre comptes et entre régions.	24 novembre 2021

Ensembles de données NoSQL Workbench au format CSV import/export	NoSQL Workbench pour Amazon DynamoDB vous permet désormais d'importer et de remplir automatiquement des exemples de données pour vous aider à créer et à visualiser vos modèles de données.	11 octobre 2021
Filtrez et récupérez l'activité du plan de données Amazon DynamoDB Streams avec AWS CloudTrail	Amazon DynamoDB vous offre désormais un contrôle plus précis de la journalisation des audits en vous permettant de filtrer l'activité Streams de l'API du plan de données dans AWS CloudTrail.	22 septembre 2021
Console mise à jour	La Console DynamoDB est désormais votre console par défaut pour vous aider à gérer les données plus facilement, à simplifier vos tâches courantes et à vous donner un accès plus rapide aux ressources et aux fonctionnalités.	25 août 2021
Le SDK DAX pour Java 2.x est désormais disponible	Le SDK DynamoDB Accelerator (DAX) pour Java 2.x est désormais disponible et est compatible avec le SDK pour Java 2.x. AWS Vous pouvez bénéficier des dernières fonctions, notamment des fonctions d'I/O non bloquantes.	29 juillet 2021

Mises à jour des fonctionnalités NoSQL Workbench, y compris les opérations du plan	NoSQL Workbench pour Amazon DynamoDB vous aide désormais à exécuter des opérations fréquentes plus facilement pour modifier les données des tables et y accéder.	28 juillet 2021
Les tables globales DynamoDB sont désormais disponibles dans la région Asie-Pacifique	Tables globales DynamoDBsont désormais disponibles dans la région Asie-Pacifique (Osaka). Les tables globales répliquent automatiquement vos tables Amazon DynamoDB dans les régions AWS de votre choix.	28 juillet 2021
DAX est désormais disponible en Chine	DynamoDB Accelerator (DAX) est désormais disponible dans la région Chine (Beijing) gérée par Sinnet.	28 juillet 2021
Chiffrement DAX en transit	DynamoDB Accelerator (DAX) prend désormais en charge le chiffrement lors du transit de données entre vos applications et clusters DAX, et entre les nœuds d'un cluster DAX.	14 juillet 2021
CloudFormation et CloudTrail intégration	Intégration AWS CloudFormation et amélioration de la sécurité grâce à la journalisation du CloudFormation plan de données.	18 juin 2021

CloudFormation désormais pris en charge pour les tables globales	Les tables globales Amazon DynamoDB sont désormais prises en AWS CloudFormationcharge, ce qui signifie que vous pouvez créer des tables globales et gérer leurs paramètres à l'aide de modèles. CloudFormation	14 mai 2021
Prise en charge locale Amazon DynamoDB pour Java 2.x	Vous pouvez maintenant utiliser l'AWS Kit SDK pour Java 2.xavecDynamoDB Local, la version télécharg eable d'Amazon DynamoDB. Avec DynamoDB local, vous pouvez développer et tester des applications à l'aide d'une version de DynamoDB exécutée dans votre environne ment de développement local sans coûts supplémentaires.	3 mai 2021
NoSQL Workbench prend désormais en charge AWS CloudFormation	NoSQL Workbench pour Amazon DynamoDB est désormais compatible AWS CloudFormation, ce qui vous permet de gérer et de modifier les modèles de données DynamoDB à l'aide de modèles. CloudFormation En outre, vous pouvez désormais configurer les paramètres de capacité de table dans NoSQL Workbench.	22 avril 2021

DynamoDB et désormais intégration des fonctionnalités AWS Amplify	AWS Amplify orchestre désormais plusieurs mises à jour d'index secondaires globaux DynamoDB dans un seul déploiement.	20 avril 2021
AWS CloudTrail pour enregistrer le plan de données Amazon DynamoDB Streams APIs	Vous pouvez désormais utiliser AWS CloudTrail pour journaliser l'activité de l'API de plan de données Amazon DynamoDB Streams, et surveiller et étudier les changements au niveau des éléments dans vos tables DynamoDB.	20 avril 2021
Amazon Kinesis Data Streams pour Amazon DynamoDB est désormais compatible AWS CloudFormation	Amazon Kinesis Data Streams pour Amazon DynamoDB est AWS CloudFormation désormais compatible, ce qui signifie que vous pouvez activer le streaming vers un flux de données Amazon Kinesis sur vos tables DynamoDB à l'aide de modèles. CloudFormation En diffusant vos modifications de données DynamoDB dans un flux de données Kinesis, vous pouvez créer des applications de streaming avancées avec les services Kinesis. AAmazon	12 avril 2021

Amazon Keyspaces propose désormais des terminaux conformes à la norme FIPS 140-2	Amazon Keyspaces (pour Apache Cassandra) propose désormais des points de terminaison conformes aux normes FIPS (Federal Information Processing Standards) 140-2 pour vous aider à exécuter plus facilement les charges de travail hautement réglementées. La publication Federal Information Processing Standard (FIPS) 140-2 est une norme de sécurité du gouvernement américain qui spécifie les exigences de sécurité pour les modules cryptographiques qui protègent des informations sensibles.	08 avril 2021
Instances Amazon EC2 T3 pour DAX	DAX prend désormais en charge les types d'instances Amazon EC2 T3, qui fournissent un niveau de performance de base du processeur avec la possibilité de dépasser le niveau de référence en cas de besoin.	15 février 2021
Prise en charge de NoSQL Workbench pour Amazon DynamoDB pour PartiQL	Vous pouvez désormais utiliser le NoSQL Workbench pour DynamoDB pour générer des instructions PartiQL pour DynamoDB.	4 décembre 2020

PartiQL pour DynamoDB	Vous pouvez désormais utiliser PartiQL pour DynamoDB, un langage de requête compatible avec SQL, pour interagir avec les tables DynamoDB et exécuter des requêtes ad hoc en utilisant le, et DynamoDB pour PartiQL. AWS Management Console AWS Command Line Interface APIs	23 novembre 2020
Amazon Kinesis Data Streams pour Amazon DynamoDB	Vous pouvez désormais utiliser Amazon Kinesis Data Streams pour Amazon DynamoDB avec vos tables DynamoDB pour récupérer les modifications au niveau de l'élément et les répliquer dans un flux de données Kinesis.	23 novembre 2020
Exportation de table DynamoDB	Vous pouvez désormais exporter vos tables DynamoDB vers Amazon S3, ce qui vous permet d'effectuer des analyses et des requêtes complexes sur vos données avec des services tels qu' AWS Glue Athena et Lake Formation.	9 novembre 2020

Prise en charge des valeurs vides	DynamoDB prend désormais en charge les valeurs vides pour les attributs String (chaîne) et Binary (binaire) autres que de clés dans les tables DynamoDB. La prise en charge des valeurs vides vous offre une plus grande flexibilité pour utiliser des attributs pour un ensemble plus large de cas d'utilisation sans avoir à transformer ces attributs avant de les envoyer à DynamoDB. Les types de données Set (ensemble), List (liste) et Map (mappage) prennent également en charge les valeurs String (chaîne) et Binary (binaire) vides.	18 mai 2020
Prise en charge de NoSQL Workbench pour Amazon DynamoDB pour Linux	NoSQL Workbench pour Amazon DynamoDB est désormais pris en charge sur Linux Ubuntu, Fedora et Debian.	4 mai 2020

CloudWatch Informations sur les contributeurs pour DynamoDB — GA	CloudWatchContributor Insights for DynamoDB est généralement disponible. CloudWatch Contributor Insights for DynamoDB est un outil de diagnostic qui fournit at-a-glance une vue des tendances de trafic de votre table DynamoDB et vous aide à identifier les touches les plus fréquemment consultée s de votre table (également appelées touches de raccourci).	2 avril 2020
Mise à niveau des tables globales	Vous pouvez désormais mettre à jour vos tables globales de la version 2017.11.29 vers la dernière version des tables globales (2019.11.21) en quelques clics dans la console DynamoDB. En mettant à niveau la version de vos tables globales, vous pouvez augmenter facilemen t la disponibilité de vos tables DynamoDB en étendant vos tables existantes à des régions AWS supplémentaires, sans qu'aucune reconstruction de table ne soit nécessaire.	16 mars 2020

NoSQL Workbench pour Amazon DynamoDB – Disponibilité générale (GA)	NoSQL Workbench pour Amazon DynamoDB est généralement disponible. Utilisez NoSQL Workbench pour concevoir, créer, interroger et gérer des tables DynamoDB.	2 mars 2020
Métriques de cluster de caches DAX	Support DAX pour de nouvelles CloudWatch métriques, qui vous permettent de mieux comprendre les performances de votre cluster DAX.	6 février 2020
CloudWatch Informations sur les contributeurs pour DynamoDB — Version préliminaire	CloudWatchContributor Insights for DynamoDB est un outil de diagnostic qui fournit at-a-glance une vue des tendances de trafic de votre table DynamoDB et vous aide à identifier les touches les plus fréquemment consultées de votre table (également appelées touches de raccourci).	26 novembre 2019

Prise en charge de capacité adaptative pour charge de travail déséquilibrée	La capacité adaptative d'Amazon DynamoDB gère désormais mieux les charges de travail déséquilibrées en isolant automatiquement les éléments fréquemment utilisés. Si votre application génère un trafic excessivement élevé vers un ou plusieurs éléments, DynamoDB rééquilibre vos partitions de manière à ce que les éléments fréquemment utilisés ne résident pas sur la même partition.	26 novembre 2019
Prise en charge des clés gérées par le client	Désormais, DynamoDB prend en charge les clés gérées par le client, ce qui signifie que vous disposez d'un contrôle total sur la façon dont vous chiffrez et gérez la sécurité de vos données DynamoDB.	25 novembre 2019
Prise en charge de NoSQL Workbench pour DynamoDB local (version téléchargeable)	NoSQL Workbench prend désormais en charge la connexion à DynamoDB local (version téléchargeable) pour concevoir, créer, interroger et gérer des tables DynamoDB.	8 novembre 2019

NoSQL Workbench - Version préliminaire	Il s'agit de la version initiale de NoSQL Workbench pour DynamoDB. Utilisez NoSQL Workbench pour concevoir , créer, interroger et gérer des tables DynamoDB. Pour de plus amples informations, veuillez consulter NoSQL Workbench pour Amazon DynamoDB (version préliminaire).	16 septembre 2019
DAX ajoute la prise en charge des opérations transactionnelles à l'aide de Python et .NET	DAX prend en charge `TransactWriteItems` et `TransactGetItems` APIs pour les applications écrites en Go, Java, .NET, Node.js et Python. Pour plus d'informations, voir Accélération en mémoire avec DAX.	14 février 2019
Mises à jour d'Amazon DynamoDB Local (version téléchargeable)	DynamoDB local (version téléchargeable) prend désormais en charge les capacités APIs transactionnelles read/write à la demande, les rapports de capacité pour les opérations de lecture et d'écriture, ainsi que 20 index secondaires globaux. Pour plus d'informations, consultez Différences entre DynamoDB téléchargeable et Amazon DynamoDB Web Service.	4 février 2019

Amazon DynamoDB à la demande	DynamoDB à la demande est une option de facturation souple permettant de servir des milliers de demandes par seconde sans planification des capacités. DynamoDB à la demande pay-per-request propose des tarifs pour les demandes de lecture et d'écriture afin que vous ne payiez que pour ce que vous utilisez. Pour plus d'informations, consultez Capacité de débit DynamoDB.	28 novembre 2018
Amazon DynamoDB Transactions	Les transactions DynamoDB apportent des modifications coordonnées all-or-nothing à plusieurs éléments à la fois dans et entre les tables, garantissant ainsi l'atomicité, la cohérence, l'isolation et la durabilité (ACID) dans DynamoDB. Pour en savoir plus, consultez Amazon DynamoDB Transactions.	27 novembre 2018

Amazon DynamoDB chiffre toutes les données client au repos	Le chiffrement DynamoDB au repos offre une couche supplémentaire de protection des données en sécurisant vos données dans la table chiffrée, y compris sa clé primaire, les index secondaires locaux et globaux, les flux, les tables globales, les sauvegardes et les clusters DAX chaque fois que les données sont stockées sur un support durable. Pour en savoir plus, consultez Chiffrement au repos Amazon DynamoDB.	15 novembre 2018
Utilisation locale d'Amazon DynamoDB facilitée avec la nouvelle image Docker	Il est désormais plus facile d'utiliser DynamoDB local, la version téléchargeable de DynamoDB, pour développer et tester vos applications DynamoDB à l'aide de la nouvelle image Docker DynamoDB local. Pour plus d'informations, consultez DynamoDB (version téléchargeable) et Docker.	22 août 2018

DynamoDB Accelerator (DAX) ajoute une prise en charge pour le chiffrement au repos	DynamoDB Accelerator (DAX) prend désormais en charge le chiffrement au repos pour les nouveaux clusters DAX pour vous aider à accélérer les lectures à partir de tables Amazon DynamoDB dans les applications sensibles en termes de sécurité, qui sont soumises à des exigences strictes en matière de conformité et de réglementation. Pour plus d'informations, consultez la section Chiffrement DAX au repos.	9 août 2018
point-in-timeDynamoDB Recovery (PITR) ajoute la prise en charge de la restauration de tables supprimées	Si vous supprimez une table avec la point-in-time restauration activée, une sauvegarde du système est automatiquement créée et conservée pendant 35 jours (sans frais supplémentaires). Pour plus d'informations, consultez Avant de commencer à utiliser la restauration à un instant dans le passé.	7 août 2018

Mises à jour disponibles sur RSS	Vous pouvez à présent vous abonner au flux RSS (dans le coin supérieur gauche de cette page) pour recevoir les notifications des mises à jour apportées au Manuel du développeur Amazon DynamoDB.	3 juillet 2018

Mises à jour antérieures

Le tableau ci-après décrit les modifications importantes apportées au Guide du développeur DynamoDB avant le 3 juillet 2018.

Modifier	Description	Date de modification
Prise en charge de Go pour DAX	Vous pouvez désormais activer des performances de lecture de l'ordre de la microseconde pour les tables Amazon DynamoDB de vos applications écrites en langage de programmation Go à l'aide du nouveau kit SDK DynamoDB Accelerator (DAX) pour Go. Pour de plus amples informations, veuillez consulter Kit SDK DAX pour Go.	26 juin 2018
DynamoDB annonce un SLA	DynamoDB a publié un contrat de niveau de service (SLA) de disponibilité publique. Pour plus d'informations, consultez le contrat de niveau de service (SLA) Amazon DynamoDB.	19 juin 2018

Modifier	Description	Date de modification
Sauvegardes Point-In-Time et restauration continues DynamoDB (PITR)	Point-in-time La restauration permet de protéger vos tables Amazon DynamoDB contre les opérations d'écriture ou de suppression accidente lles. Grâce à la restauration à un instant dans le passé, vous n'avez plus à vous soucier de la création, de la maintenance ou de la planifica tion des sauvegardes à la demande. Par exemple, imaginez qu'un script de test écrive accidentellement sur une table DynamoDB de production. Grâce à point-in-time la restauration, vous pouvez restaurer cette table à n'importe quel moment au cours des 35 derniers jours. DynamoDB conserve des sauvegardes incrémentielles de votre table. Pour de plus amples informations, veuillez consulter Sauvegardes ponctuelles pour DynamoDB.	25 avril 2018

Modifier	Description	Date de modification
Chiffrement au repos pour DynamoDB	Le chiffrement au repos DynamoDB, disponible pour les nouvelles tables DynamoDB, vous aide à sécuriser vos données d'application dans les tables Amazon DynamoDB en utilisant des clés de chiffrement gérées par AWS stockées dans AWS Key Management Service. Pour de plus amples informations, veuillez consulter Chiffrement de DynamoDB au repos.	8 février 2018

Modifier	Description	Date de modification
Sauvegarde et restauration de DynamoDB	La sauvegarde à la demande vous permet de créer des sauvegardes complètes des données de vos tables DynamoDB pour l'archiva ge des données, ce qui vous permet de répondre aux exigences réglementaires de votre entreprise et du gouvernement. Vous pouvez sauvegarder des tables de quelques méga-octets à des centaines de téra-octets de données, sans impact sur les performances et la disponibi lité de vos applications de production. Pour de plus amples informations, veuillez consulter Sauvegarde et restauration pour DynamoDB.	29 novembre 2017

Modifier	Description	Date de modification
Tables globales DynamoDB	Les tables globales s'appuient sur l'étendue internationale de DynamoDB pour vous fournir une base de données entièrement gérée, à plusieurs régions et à multiples activités qui fournit des performances de lecture et d'écriture rapides et locales, pour des applications dimensionnées massivement et internationales. Global Tables réplique automatiquement vos tables Amazon DynamoDB dans les régions de votre choix. AWS Pour de plus amples informations, veuillez consulter Tables globales : réplication multi-active, multirégion.	29 novembre 2017
Prise en charge de Node.js pour DAX	Les développeurs Node.js peuvent utiliser DynamoDB Accelerator (DAX), à l'aide du client DAX pour Node.js. Pour de plus amples informations, veuillez consulter Accélération en mémoire avec DynamoDB Accelerator (DAX).	5 octobre 2017

Modifier	Description	Date de modification
Points de terminaison de VPC pour DynamoDB	Les points de terminaison DynamoDB permettent aux EC2 instances Amazon de votre Amazon VPC d'accéder à DynamoDB, sans accès à Internet public. Le trafic réseau entre votre VPC et DynamoDB ne quitte pas le réseau Amazon. Pour de plus amples informations, veuillez consulter Utilisation de points de terminaison d'un VPC Amazon pour accéder à DynamoDB.	le 16 août 2017

Modifier	Description	Date de modification
Scalabilité automatique pour DynamoDB	La scalabilité automatique DynamoDB évite de devoir définir ou ajuster manuellement les paramètres de débit provisionné. En effet, la scalabilité automatique DynamoDB ajuste de façon dynamique la capacité en écriture et en lecture, en réponse aux modèles de trafic réels. Cela permet à une table ou à un index secondaire global d'augmenter les capacités en lecture et écriture qui lui sont allouées afin de gérer les hausses soudaines de trafic sans limitation. Lorsque la charge de travail diminue, la scalabilité automatique DynamoDB réduit la capacité provisionnée. Pour de plus amples informations, veuillez consulter Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.	14 juin 2017

Modifier	Description	Date de modification
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) est un service de mise en cache en mémoire entièrement géré et hautement disponible pour DynamoDB. Il offre des performances jusqu'à 10 fois supérieur es, de l'ordre de quelques microsecondes au lieu de quelques millisecondes, même lorsque le nombre de requêtes s'élève à plusieurs millions par seconde. Pour de plus amples informations, veuillez consulter Accélération en mémoire avec DynamoDB Accelerator (DAX).	19 avril 2017
DynamoDB prend désormais en charge l'expiration automatique des éléments avec le time-to-live (TTL)	Le time-to-live (TTL) d'Amazon DynamoDB vous permet de supprimer automatiquement de vos tables les éléments ayant expiré, sans frais supplémen taires. Pour de plus amples informations, veuillez consulter Utilisation de la durée de vie (TTL) dans DynamoDB.	27 février 2017

Modifier	Description	Date de modification
DynamoDB prend désormais en charge les étiquettes de répartition des coûts	Vous pouvez désormais ajouter des balises à vos tables Amazon DynamoDB afin de profiter d'un meilleur classement par catégorie de l'utilisation et de rapports plus précis sur les coûts. Pour de plus amples informations, veuillez consulter Ajout de balises et d'étiquettes aux ressources dans DynamoDB.	19 janvier 2017

Modifier	Description	Date de modification
Nouvelle API DynamoDB `DescribeLimits`	L'`DescribeLimits` API renvoie les limites de capacité actuellement allouées pour votre AWS compte dans une région, à la fois pour la région dans son ensemble et pour toute table DynamoDB que vous y créez. Elle vous permet de déterminer quelles sont vos limites actuelles au niveau du compte afin que vous puissiez les comparer à la capacité allouée que vous utilisez actuellement, et que vous disposiez de tout le temps nécessaire pour demander une augmentation avant d'atteindre une limite. Pour plus d'informations, consultez Quotas dans Amazon DynamoDB et consultez le DescribeLimitsmanuel Amazon DynamoDB API Reference.	1er mars 2016

Modifier	Description	Date de modification
Mise à jour de la console DynamoDB et nouvelle terminologie pour les attributs de clé primaire	La console de gestion DynamoDB a été repensée pour être plus intuitive et plus facile à utiliser. Dans le cadre de cette mise à jour, nous vous présentons la nouvelle terminologie pour les attributs de clé primaires : • Clé de partition – Également appelée attribut de hachage. • Clé de tri – Également appelée attribut de plage. Seuls les noms ont changé ; la fonctionnalité demeure la même. Lorsque vous créez une table ou un index secondaire, vous pouvez choisir une clé primaire simple (clé de partition uniquement) ou une clé primaire composite (clé de partition et clé de tri). La documentation DynamoDB a été mise à jour pour refléter ces modifications.	le 12 novembre 2015

Modifier	Description	Date de modification
Backend de stockage Amazon DynamoDB pour Titan	Le backend de stockage DynamoDB pour Titan est un backend de stockage pour la base de données de graphiques Titan implémentée sur Amazon DynamoDB. Lorsque vous utilisez le backend de stockage DynamoDB pour Titan, vos données bénéficient de la protection de DynamoDB, qui s'exécute dans les centres de données haute disponibilité d'Amazon. Le plug-in est disponible pour Titan version 0.4.4 (principalement pour la compatibilité avec des applications existantes) et Titan version 0.5.4 (recommandé pour les nouvelles applications). Comme les autres backend de stockage pour Titan, ce plug-in prend en charge la pile Tinkerpop (versions 2.4 et 2.5), y compris l'API Blueprints et le shell Gremlin.	20 août 2015

Modifier	Description	Date de modification
DynamoDB Streams, réplication entre régions et analyse avec des lectures cohérentes fortes	DynamoDB Streams capture une séquence chronologique des modifications au niveau des éléments dans toute table DynamoDB, et stocke ces informations dans un journal pendant jusqu'à 24 heures. Les applications ont accès à ce journal et affichent les éléments de données à mesure qu'ils s'affichent avant et après qu'ils ont été modifiés, pratiquement en temps réel. Pour plus d'informations, consultez Modifier la récupération de données pour DynamoDB Streams et la Référence d'API DynamoDB Streams. La réplication entre régions DynamoDB est une solution côté client qui permet de conserver des copies identiques des tables DynamoDB dans différentes régions, quasiment en temps réel. AWS Vous pouvez utiliser la réplication entre régions pour sauvegarder les tables DynamoDB ou pour fournir un accès à faible latence aux données là où les utilisateurs sont répartis géographiquement.	16 juillet 2015

Modifier	Description	Date de modification
	L'opération DynamoDB `Scan` utilise par défaut des lectures éventuellement cohérente s. Vous pouvez utiliser des lectures cohérentes fortes au lieu de configurer le paramètre `ConsistentRead` sur true. Pour plus d'informa tions, consultez Cohérence en lecture de l'opération d'analyse et Analyser dans la Référence d'API Amazon DynamoDB.	
AWS CloudTrail support pour Amazon DynamoDB	DynamoDB est désormais intégré à. CloudTrail CloudTrai l capture les appels d'API effectués depuis la console DynamoDB ou depuis l'API DynamoDB et les suit dans des fichiers journaux. Pour plus d'informations, consultez Journalisation des opérations de DynamoDB à l'aide d'AWS CloudTrail et le Guide de l'utilisateur AWS CloudTrail.	28 mai 2015

Modifier	Description	Date de modification
Amélioration du support pour les expressions de requête	Cette version ajoute un nouveau paramètre `KeyConditionExpression` à l'API `Query`. Une `Query` lit les éléments d'une table ou d'un index à l'aide de valeurs de clé primaire. Le paramètre `KeyConditionExpression` est une chaîne qui identifie les noms de clé primaire et les conditions à appliquer aux valeurs de clé. La `Query` récupère uniquement les éléments qui satisfont l'expression. La syntaxe de `KeyConditionExpression` est similaire à celle d'autres paramètres d'expression dans DynamoDB. Elle vous permet de définir des variables de substitution pour des noms et des valeurs dans l'expression. Pour de plus amples informations, veuillez consulter Interrogation de tables dans DynamoDB.	27 avril 2015

Modifier	Description	Date de modification
Nouvelles fonctions de comparaison pour écritures conditionnelles	Dans DynamoDB, le paramètre `Condition Expression` détermine si une opération `PutItem`, `UpdateItem` ou `DeleteItem` aboutit. L'élément n'est écrit que si le résultat de la condition est true (vrai). Cette version ajoute deux nouvelles fonctions, `attribute_type` et `size`, à utiliser avec `ConditionExpression`. Ces fonctions vous permettent d'effectuer des écritures conditionnelles basées sur le type de données ou la taille d'un attribut dans une table. Pour de plus amples informations, veuillez consulter Exemple de commande CLI d'expression de condition DynamoDB.	27 avril 2015

Modifier	Description	Date de modification
Analyse d'API pour index secondaires	Dans DynamoDB, une opération `Scan` lit tous les éléments dans une table, applique des critères de filtrage définis par l'utilisateur et renvoie les éléments de données sélectionnés à l'application. Cette même fonctionnalité est désormais disponible pour les index secondaires également. Pour rechercher dans un index local secondaire ou un index global secondaire, vous spécifiez le nom de l'index et le nom de sa table parent. Par défaut, un index `Scan` renvoie toutes les données dans l'index. Vous pouvez utiliser une expression de filtre pour affiner les résultats qui sont renvoyés à l'application. Pour de plus amples informations, veuillez consulter Analyse de tables dans DynamoDB.	10 février 2015

Modifier	Description	Date de modification
Opérations en ligne pour index globaux secondaires	L'indexation en ligne vous permet d'ajouter ou de supprimer des index globaux secondaires sur des tables existantes. Avec l'indexation en ligne, vous n'avez pas besoin de définir tous les index d'une table lorsque vous créez une table. Au lieu de cela, vous pouvez ajouter un nouvel index à tout moment. De même, si vous jugez que vous n'avez plus besoin d'un index, vous pouvez le supprimer à tout moment. Les opérations d'indexation en ligne ne sont pas bloquantes. Ainsi, la table reste disponible pour des activités de lecture et d'écriture pendant que des index sont ajoutés ou supprimés. Pour de plus amples informations, veuillez consulter Gestion des index secondaires globaux dans DynamoDB.	27 janvier 2015

Modifier	Description	Date de modification
Prise en charge de modèle de document avec JSON	DynamoDB vous permet de stocker et de récupérer des documents avec une prise en charge totale des modèles de document. Les nouveaux types de données sont entièrement compatibles avec la norme JSON et vous permettent d'imbriquer des éléments de document les uns dans les autres. Vous pouvez utiliser les opérateurs de déréférençage de chemin d'accès du document afin de lire et d'écrire des éléments individuels, sans avoir besoin de récupérer l'ensemble du document. Cette version présente également de nouveaux paramètres d'expression pour spécifier des projections, des conditions et des actions de mise à jour lors de la lecture ou de l'écriture d'éléments de données. Pour en savoir plus sur la prise en charge du modèle de document avec JSON, consultez Types de données et Utilisation d'expressions dans DynamoDB.	7 octobre 2014

Modifier	Description	Date de modification
Mise à l'échelle flexible	Pour les tables et les index globaux secondaires, vous pouvez accroître la capacité allouée de débit de lecture et d'écriture de n'importe quelle quantité, à condition de rester dans vos limites par table et par compte. Pour de plus amples informations, veuillez consulter Quotas dans Amazon DynamoDB.	7 octobre 2014
Tailles d'éléments supérieures	La taille d'élément maximum dans DynamoDB a augmenté, passant de 64 Ko à 400 Ko. Pour de plus amples informations, veuillez consulter Quotas dans Amazon DynamoDB.	7 octobre 2014

Modifier	Description	Date de modification
Expressions conditionnelles améliorées	DynamoDB étend les opérateurs disponibles pour les expressions condition nelles. Vous disposez ainsi de davantage de flexibili té pour des insertions, des mises à jour et des suppressi ons conditionnelles. Les opérateurs nouvellement disponibles vous permetten t de contrôler si un attribut existe ou non, est supérieur ou inférieur à une valeur particuli ère, est entre deux valeurs, commence par certains caractères et bien plus encore. DynamoDB fournit également un opérateur OR (OU) pour évaluer plusieurs condition s. Par défaut, plusieurs conditions dans une expressio n sont reliées par AND afin que l'expression soit true uniquement si toutes ses conditions sont true. Si vous spécifiez OR à la place, l'expression est true si une ou plusieurs conditions sont remplies. Pour de plus amples informations, veuillez consulter Utilisation d'éléments et d'attributs dans DynamoDB.	24 avril 2014

Modifier	Description	Date de modification
Filtre de requête	L'API DynamoDB `Query` prend en charge une nouvelle option `QueryFilter` . Par défaut, une `Query` recherche des éléments qui correspon dent à une valeur de clé de partition spécifique et une condition de clé de tri en option. Un filtre de `Query` applique des expressions conditionnelles aux autres attributs non-clés. Si un filtre `Query` est présent, alors les éléments qui ne correspon dent pas aux conditions de filtre sont ignorées avant que les résultats de la `Query` soient renvoyés à l'applica tion. Pour de plus amples informations, veuillez consulter Interrogation de tables dans DynamoDB.	24 avril 2014

Modifier	Description	Date de modification
Exportation et importation de données à l'aide du AWS Management Console	La console DynamoDB a été améliorée pour simplifie r les exportations et les importations de données dans des tables DynamoDB. En quelques clics, vous pouvez configurer et AWS Data Pipeline orchestrer le flux de travail, ainsi qu'un MapReduce cluster Amazon Elastic pour copier les données des tables DynamoDB vers un compartim ent Amazon S3, ou vice-vers a. Vous pouvez effectuer une exportation ou une importati on une seule fois, ou configure r une tâche d’exportation quotidienne. Vous pouvez même effectuer des exportati ons et des importations entre régions, en copiant les données DynamoDB d'une table d'une région vers une table d' AWS une autre région. AWS	6 mars 2014

Modifier	Description	Date de modification
Documentation d'API de niveau plus élevé réorganisée	Les informations sur les sujets suivants APIs sont désormais plus faciles à trouver : • Java : Dynamo DBMappper • .NET : modèle de document et modèle de persistance des objets Ces niveaux supérieurs APIs sont désormais documentés ici :. Interfaces de programmation de niveau supérieur pour DynamoDB	20 janvier 2014

Modifier	Description	Date de modification
Index secondaires globaux	DynamoDB prend en charge les index secondaires globaux. De la même manière qu'avec un index secondaire local, vous définissez un index secondaire global à l'aide d'une clé secondaire à partir d'une table puis en émettant des demandes de requête sur l'index. Contrairement à un index secondaire local, la clé de partition pour l'index secondaire global n'a pas à être identique à celle de la table. Elle peut être n'importe quel attribut scalaire de la table. La clé de tri est facultative et peut également être n'importe quel attribut scalaire de table. Un index secondaire global a également ses propres paramètres de débit alloués qui sont distincts de ceux de la table parent. Pour plus d'informations, consultez Amélioration de l'accès aux données avec les index secondaires dans DynamoDB et Utilisation d'index secondaires globaux dans DynamoDB.	12 décembre 2013

Modifier	Description	Date de modification
Contrôle précis des accès	DynamoDB ajoute une prise en charge d'un contrôle précis des accès. Cette fonction permet aux clients de spécifier les principaux (utilisateurs, groupes ou rôles) pouvant accéder à des attributs et éléments individuels dans une table DynamoDB ou un index secondaire. Les applications peuvent également tirer parti de la fédération des identités web pour décharger la tâche d'authentification utilisateur à un fournisseur d'identité de tiers, tel que Facebook, Google ou Login with Amazon. De cette façon, les applicati ons (y compris les applicati ons mobiles) peuvent gérer un très grand nombre d'utilisa teurs, tout en s'assurant que personne ne puisse accéder aux éléments de données DynamoDB sans y être autorisé. Pour de plus amples informations, veuillez consulter Utilisation de conditions de politique IAM pour un contrôle d'accès précis.	29 octobre 2013

Modifier	Description	Date de modification
Taille d'unité de capacité de lecture de 4 Ko	La taille d'unité de capacité pour les lectures a augmenté, passant de 1 Ko à 4 Ko. Cette amélioration peut réduire le nombre d'unités de capacité de lecture fournies requises pour de nombreuses applications. Par exemple, avant cette version, la lecture d'un élément de 10 Ko utilisait 10 unités de capacité de lecture. Maintenant, ces mêmes 10 Ko lus utiliseraient seulement 3 unités (10 Ko/4 Ko, arrondis à la prochaine limite de 4 Ko). Pour de plus amples informations, veuillez consulter Capacité de débit DynamoDB.	14 mai 2013

Modifier	Description	Date de modification
Analyses parallèles	DynamoDB ajoute la prise en charge des opération s d'analyse parallèles. Les applications peuvent maintenant diviser une table en segments logiques et analyser tous les segments simultanément. Cette fonctionnalité réduit le temps nécessaire pour mener une analyse complète et utilise pleinement la capacité de lecture allouée de la table. Pour de plus amples informati ons, veuillez consulter Analyse de tables dans DynamoDB.	14 mai 2013
Index locaux secondaires	DynamoDB ajoute la prise en charge des index secondair es locaux. Vous pouvez définir des index de clé de tri sur des attributs non-clés, puis utiliser ces index dans des demandes d'interrogation. Avec des index secondair es locaux, les applications peuvent extraire efficacement des éléments de données sur plusieurs dimensions. Pour de plus amples informations, veuillez consulter Index locaux secondaires.	18 avril 2013

Modifier	Description	Date de modification
Nouvelle version d'API	Avec cette version, DynamoDB introduit une nouvelle version d'API (2012-08-10). La version précédente d'API (2011-12-05) est toujours prise en charge pour la compatibilité descendante avec des applications existantes. Les nouvelles applications doivent utiliser la nouvelle version d'API (2012-08-10). Nous vous recommandons de migrer vos applications existantes vers la version d'API 2012-08-10. En effet, les nouvelles fonctions DynamoDB (par exemple, les index locaux secondaires) ne sont pas rétroportées vers la version d'API précédente. Pour plus d'informations sur l'API version 2012-08-10, consultez la Référence d'API Amazon DynamoDB API.	18 avril 2013

Modifier	Description	Date de modification
Prise en charge des variables dans la politique IAM	Le langage de la politique d'accès IAM prend désormais en charge des variables . Lorsqu'une politique est évaluée, toutes les variables de politique sont remplacée s par des valeurs issues d'informations basées sur le contexte provenant de la session de l'utilisateur authentifié. Vous pouvez utiliser des variables de politique pour définir des politiques à usage général sans dresser de manière explicite la liste des éléments composant cette politique . Pour plus d'informations sur les variables de politique , consultez Policy Variables (Variables de politique) dans le guide IAM Utilisation d'Gestion des identités et des accès AWS . Pour obtenir des exemples de variables de politique dans DynamoDB, consultez Gestion des identités et des accès pour Amazon DynamoDB.	4 avril 2013

Modifier	Description	Date de modification
Exemples de code PHP mis à jour pour AWS SDK pour PHP la version 2	La version 2 AWS SDK pour PHP est désormais disponibl e. Les exemples de code PHP dans le Guide du développe ur Amazon DynamoDB ont été mis à jour pour utiliser ce nouveau kit SDK. Pour plus d'informations sur la version 2 du kit SDK, consultez AWS SDK pour PHP.	23 janvier 2013
Nouveau point de terminaison	DynamoDB s'étend à la région (ouest AWS GovCloud des États-Unis). Pour la liste actuelle des points de terminaison de service et des protocoles, consultez Régions et points de terminaison.	3 décembre 2012
Nouveau point de terminaison	DynamoDB s'étend à la région Amérique du Sud (Sao Paulo). Pour obtenir la liste actuelle des points de terminaison pris en charge, consultez Régions et points de terminaison.	3 décembre 2012
Nouveau point de terminaison	DynamoDB s'étend à la région Asie-Pacifique (Sydney). Pour obtenir la liste actuelle des points de terminaison pris en charge, consultez Régions et points de terminaison.	13 novembre 2012

Modifier	Description	Date de modification
DynamoDB implément e la prise en charge des checksums, prend en charge CRC32 les extractions par lots très cohérentes et supprime les restrictions relatives aux mises à jour simultanées des tables.	• DynamoDB calcule CRC32 une somme de contrôle de la charge utile HTTP et renvoie cette somme de contrôle dans un nouvel en-tête,. `x-amz-crc32` Pour de plus amples informati ons, veuillez consulter API de bas niveau de DynamoDB. • Par défaut, les opération s de lecture effectuée s par l'API `BatchGetI tem` sont éventuellement cohérentes. Un nouveau paramètre `Consisten tRead` dans `BatchGetI tem` vous permet de choisir au lieu de cela une cohérence de lecture forte, pour toutes les tables dans la demande. Pour de plus amples informations, veuillez consulter Descripti on. • Cette version supprime certaines restrictions lors de la mise à jour de nombreuse s tables simultanément. Le nombre total de tables qui peuvent être mises à jour simultanément est toujours de 10. Toutefois , ces tables peuvent	2 novembre 2012

Modifier	Description	Date de modification
	maintenant correspondre à toute combinaison d'état `CREATING`, `UPDATING` ou `DELETING`. De plus, il n'y a plus de montant minimum pour augmenter ou réduire le ReadCapacityUnitsou WriteCapacityUnitspour une table. Pour de plus amples informations, veuillez consulter Quotas dans Amazon DynamoDB.	
Documentation de bonnes pratiques	Le Guide du développeur Amazon DynamoDB identifie les bonnes pratiques pour travailler avec des tables et des éléments, ainsi que des recommandations pour les opérations de requête et d'analyse.	28 septembre 2012

Modifier	Description	Date de modification
Prise en charge pour le type de données binaires	Outre les types de données Number (nombre) et String (chaîne), DynamoDB prend désormais en charge le type de données Binary (binaire). Avant cette version, pour stocker des données binaires, vous convertissiez vos données binaires au format chaîne et les stockiez dans DynamoDB. Outre le travail de conversion obligatoire sur le côté client, la conversion a souvent augmenté la taille de l'élément de données nécessitant davantage de stockage et potentiellement une capacité de débit alloué supplémentaire. Avec les attributs de type binaire, vous pouvez à présent stocker n'importe quelle donnée binaire. Par exemple, des données compressé es, des données chiffrées et des images. Pour de plus amples informations, veuillez consulter Types de données. Pour des exemples pratiques de gestion de données de type binaire à l'aide de AWS SDKs, consultez les sections suivantes :	21 août 2012

Modifier	Description	Date de modification
	• Exemple : gestion d'attributs de type binaire à l'aide de l'API de document de AWS SDK pour Java • Exemple : gestion d'attributs de type binaire à l'aide de l'API de bas niveau AWS SDK pour .NET Pour ajouter la prise en charge des types de données binaires dans le AWS SDKs, vous devrez télécharger la dernière version SDKs et vous devrez peut-être également mettre à jour les applications existantes. Pour plus d'informations sur le téléchargement de l' AWS SDKs, consultez Exemples de code .NET.	
Les éléments de table DynamoDB peuvent être mis à jour et copiés à l'aide de la console DynamoDB	Les utilisateurs DynamoDB peuvent désormais mettre à jour et copier des éléments de table à l'aide de la console DynamoDB, en plus d'être en mesure d'ajouter et de supprimer des éléments. Cette nouvelle fonctionnalité simplifie l'apport de modifications à des éléments individuels via la Console.	14 août 2012

Modifier	Description	Date de modification
DynamoDB réduit les exigences de débit de table minimum	DynamoDB prend désormais en charge des exigences de débit de table minimum, plus précisément 1 unité de capacité d'écriture et 1 unité de capacité de lecture. Pour plus d'informations, consultez la rubrique Quotas dans Amazon DynamoDB dans le Guide du développeur Amazon DynamoDB.	9 août 2012
Prise en charge signature Version 4	DynamoDB prend désormais en charge Signature Version 4 pour l'authentification des demandes.	5 juillet 2012
Prise en charge de l'explora teur de table dans la console DynamoDB	La console DynamoDB prend désormais en charge un explorateur de table qui vous permet de parcourir et d'interroger les données dans vos tables. Vous pouvez également insérer de nouveaux éléments ou supprimer des éléments existants. Les sections `SampleData` et Utilisation de la console ont été mises à jour pour ces fonctions.	22 mai 2012

Modifier	Description	Date de modification
Nouveaux points de terminaison	La disponibilité de DynamoDB augmente avec de nouveaux points de terminaison dans les régions USA Ouest (Californie du Nord), USA Ouest (Oregon) et Asie-Pacifique (Singapour). Pour obtenir la liste actuelle des points de terminaison pris en charge, accédez à Régions et points de terminaison.	24 avril 2012
BatchWriteItem Support de l'API	DynamoDB prend désormais en charge une API d'écriture par lots qui vous permet d'insérer et de supprimer plusieurs éléments à partir d'une ou plusieurs tables dans un seul appel d'API. Pour plus d'informations sur l'API d'écriture de traitement par lots DynamoDB, consultez BatchWriteItem. Pour plus d'informations sur l'utilisation des éléments et sur l'utilisation de la fonction d'écriture par lots à l'aide de AWS SDKs, reportez-vous Utilisation d'éléments et d'attributs dans DynamoDB aux sections etExemples de code .NET.	19 avril 2012

Modifier	Description	Date de modification
Plus de codes d'erreur documentés	Pour de plus amples informations, veuillez consulter Gestion des erreurs avec DynamoDB.	5 avril 2012
Nouveau point de terminaison	DynamoDB s'étend à la région Asie-Pacifique (Tokyo). Pour obtenir la liste actuelle des points de terminaison pris en charge, consultez Régions et points de terminaison.	29 février 2012
Métrique `ReturnedItemCount` ajoutée	Une nouvelle métrique, `ReturnedItemCount`, indique le nombre d'éléments renvoyés en réponse à une opération de requête ou de numérisation pour que DynamoDB soit disponible pour la surveillance. CloudWatch	24 février 2012
Exemples ajoutés pour incrémenter des valeurs	DynamoDB prend en charge l'incrémentation et la décrémentation de valeurs numériques existantes. Les exemples illustrent l'ajout à des valeurs existantes dans les sections « Mise à jour un élément » à l'emplacement suivant : Utilisation d'éléments : Java. Utilisation d'élements : .NET.	25 janvier 2012

Modifier	Description	Date de modification
Version de produit initiale	DynamoDB est présenté comme un nouveau service en version bêta.	18 janvier 2012

Fonctionnalités héritées de DynamoDB

Les rubriques suivantes concernent des fonctionnalités héritées que DynamoDB prend toujours en charge. Aucun développement actif n'est effectué sur ces fonctionnalités.

Rubriques

- Tables globales de la version 2017.11.29 (héritée)
- Version précédente de l'API DynamoDB de bas niveau (2011-12-05)
- Paramètres conditionnels hérités de DynamoDB

Tables globales de la version 2017.11.29 (héritée)

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
> Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Rubriques

- Fonctionnement des tables globales
- Bonnes pratiques et exigences pour la gestion des tables globales
- Création d'une table globale
- Surveillance des tables globales
- Utilisation d'IAM avec des tables globales

Fonctionnement des tables globales

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
> Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Les sections suivantes vous aideront à comprendre les concepts et le comportement des tables globales dans Amazon DynamoDB.

Concepts de table globaux pour la version 2017.11.29 (héritée)

Une table globale est une collection d'une ou plusieurs tables de réplique, toutes détenues par un seul compte AWS.

Une table de réplique (ou réplica) est une table DynamoDB unique qui fonctionne comme une partie d'une table globale. Chaque réplica stocke le même ensemble d'éléments de données. Une table globale donnée ne peut avoir qu'une seule table de réplique par région AWS.

Voici une présentation conceptuelle de la création d'une table globale.

1. Créez une table DynamoDB ordinaire, avec DynamoDB Streams activé, dans une région AWS.
2. Répétez l'étape 1 pour toute autre région dans laquelle vous souhaitez répliquer vos données.
3. Définissez une table globale DynamoDB basée sur les tables que vous avez créées.

L'AWS Management Console automatisant ces tâches, vous pouvez créer une table globale plus rapidement et plus facilement. Pour plus d'informations, consultez Création d'une table globale.

La table globale DynamoDB ainsi obtenue se compose de plusieurs tables de réplique, une par région, que DynamoDB traite comme une seule unité. Les réplicas a les mêmes nom de table et schéma de clé primaire. Quand une application écrit des données dans une table de réplique dans

une région, DynamoDB propage automatiquement l'écriture aux autres tables de réplique dans les autres régions AWS.

> **Important**
>
> Pour assurer la synchronisation de vos données de table, les tables globales créent automatiquement les attributs suivants pour chaque élément :
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> Ne modifiez pas ces attributs et ne créez pas d'attributs du même nom.

Vous pouvez ajouter des tables de réplique à la table globale de façon que celle-ci soit disponible dans d'autres régions. (Pour ce faire, la table globale doit être vide. En d'autres termes, aucune des tables de réplique ne peut contenir de données.)

Vous pouvez également supprimer une table de réplique d'une table globale. Si vous faites cela, la table est complètement dissociée de la table globale. Cette table nouvellement indépendante n'interagit plus avec la table globale, et les données ne sont plus propagées vers ou depuis la table globale.

> **Warning**
>
> Gardez à l'esprit que la suppression d'un réplica n'est pas un processus atomique. Pour garantir un comportement cohérent et un état connu, vous pouvez envisager de détourner le trafic d'écriture de votre application du réplica pour qu'il soit supprimé à l'avance. Après l'avoir supprimé, attendez que tous les points de terminaison de région de réplica affichent le réplica comme étant dissocié avant d'y effectuer d'autres écritures en tant que tableau des régions isolé.

Tâches courantes

Les tâches courantes pour les tables globales fonctionnent comme suit.

Vous pouvez supprimer la table de réplica d'une table globale de la même manière qu'une table normale. Cela arrêtera la réplication vers cette région et supprimera la copie de la table conservée dans cette région. Vous ne pouvez pas séparer la réplication et créer des copies de la table en tant qu'entités indépendantes.

> **Note**
>
> Vous ne pourrez supprimer une table source qu'au moins 24 heures après son utilisation pour créer une nouvelle région. Si vous essayez de la supprimer trop tôt, vous recevrez une erreur.

Des conflits peuvent apparaître si des applications mettent à jour le même élément dans différentes régions presque simultanément. Pour garantir la cohérence finale, les tables globales DynamoDB utilisent une méthode « priorité à la dernière écriture » pour rapprocher des mises à jour simultanées. Tous les réplicas s'accordent sur la dernière mise à jour et convergent vers un état dans lequel ils ont tous des données identiques.

> **Note**
>
> Il existe plusieurs méthodes pour éviter les conflits, notamment :
>
> - Utiliser une politique IAM pour n'autoriser les écritures que dans la table d'une seule région.
> - Utiliser une politique IAM pour acheminer les utilisateurs vers une seule région et laisser l'autre en veille inactive, ou acheminer les utilisateurs impairs vers une région et les utilisateurs pairs vers une autre région.
> - Éviter l'utilisation de mises à jour non idempotentes telles que Bookmark = Bookmark + 1, au profit de mises à jour statiques telles que Bookmark=25.

Surveillance des tables globales

Vous pouvez utiliser CloudWatch pour observer la métrique `ReplicationLatency`. Cette métrique suit le temps écoulé entre le moment où un élément mis à jour apparaît dans le flux DynamoDB pour une table de réplique, et le moment où il apparaît dans un autre réplica dans la table globale. La valeur `ReplicationLatency` est exprimée en millisecondes et émise pour chaque paire région-source/région-destination. Il s'agit de la seule métrique CloudWatch fournie par Global Tables v2.

Les latences que vous observerez dépendent de la distance entre les régions que vous avez choisies, ainsi que d'autres variables. Les latences comprises entre 0,5 et 2,5 secondes pour les régions peuvent être courantes dans la même zone géographique.

Durée de vie (TTL)

Vous pouvez utiliser la Durée de vie (TTL) pour spécifier un nom d'attribut dont la valeur indique l'heure d'expiration de l'article. Cette valeur est spécifiée sous forme de nombre en secondes depuis l'"époque de l'ère Unix.

Avec version héritée des tables globales, les suppressions TTL ne sont pas automatiquement répliquées sur d'autres répliques. Lorsqu'un élément est supprimé à l'aide d'une règle de durée de vie, ce travail est effectué sans utiliser d'unités d'écriture.

Notez que si la capacité d'écriture provisionnée des tables sources et cibles est très faible, cela peut entraîner une limitation, car les suppressions TTL nécessitent une capacité d'écriture.

Flux et transactions avec des transactions globales

Chaque table globale produit un flux indépendant basé sur toutes ses écritures, quel que soit le point d'origine de ces écritures. Vous pouvez choisir de consommer ce flux DynamoDB dans une région ou dans toutes les régions indépendamment.

Si vous souhaitez des écritures locales mais pas répliquées, vous pouvez ajouter votre propre attribut de région à chaque élément. Vous pouvez ensuite utiliser un filtre d'événements Lambda pour appeler uniquement le Lambda pour les écritures dans la région locale.

Les opérations transactionnelles offrent des garanties ACID (atomicité, cohérence, isolation et durabilité) UNIQUEMENT dans la région dans laquelle a été effectuée l'écriture d'origine. Les transactions des tables globales ne sont pas prises en charge dans toutes les régions.

Par exemple, si vous avez une table globale avec des réplicas dans les régions USA Est (Ohio) et USA Ouest (Oregon), et que vous réalisez une opération TransactWriteItems dans la région USA Est (Ohio), vous remarquerez peut-être des transactions partiellement incomplètes dans la région USA Ouest (Oregon) lorsque les changements sont répliqués. Les changements seront uniquement répliqués aux autres régions une fois validés dans la région source.

> **Note**
>
> - Les tables globales « contournent » DynamoDB Accelerator en mettant directement à jour DynamoDB. Par conséquent, DAX ne se rendra pas compte qu'il contient des données périmées. Le cache DAX ne sera actualisé que lorsque le TTL du cache expirera.
> - Les balises des tables globales ne se propagent pas automatiquement.

Débit de lecture et d'écriture

Les tables globales gèrent le débit de lecture et d'écriture de la manière suivante.

- La capacité d'écriture doit être identique sur toutes les instances de table dans toutes les régions.
- Avec la version 2019.11.21 (actuelle), si le tableau est configuré pour prendre en charge l'autoscaling ou est en mode à la demande, la capacité d'écriture est automatiquement synchronisée. La capacité d'écriture actuellement allouée dans chaque région augmentera et diminuera indépendamment de ces paramètres d'autoscaling synchronisés. Si la table est mise en mode à la demande, ce mode sera synchronisé avec les autres répliques.
- La capacité de lecture peut varier d'une région à l'autre, car les lectures peuvent ne pas être égales. Lorsque vous ajoutez une réplique globale à une table, la capacité de la région source est propagée. Après la création, vous pouvez ajuster la capacité de lecture d'une réplique. Ce nouveau paramètre n'est pas transféré de l'autre côté.

Cohérence et résolution des conflits

Toute modification apportée à un élément d'une table de réplique est répliquée dans tous les autres réplicas au sein de la même table globale. Dans une table globale, un élément nouvellement écrit est généralement propagé à toutes les tables de réplique en quelques secondes.

Avec une table globale, chaque table de réplique stocke le même ensemble d'éléments de données. DynamoDB ne prend pas en charge une réplication partielle limitée à certains éléments.

Une application peut lire et écrire des données dans n'importe quelle table de réplique. DynamoDB prend en charge les lectures éventuellement cohérentes entre régions, mais pas celles fortement cohérentes entre régions. Si votre application utilise uniquement des lectures éventuellement cohérentes et n'émet de lectures que sur une région AWS, elle fonctionnera sans aucune

modification. En revanche, si votre application nécessite des lectures fortement cohérentes, elle doit effectuer toutes les lectures et écritures fortement cohérentes dans la même région. Autrement, si vous écrivez dans une région et lisez dans une autre, la réponse de lecture peut inclure des données obsolètes ne reflétant pas les résultats des écritures récentes dans l'autre région.

Des conflits peuvent apparaître si des applications mettent à jour le même élément dans différentes régions presque simultanément. Pour veiller à la cohérence éventuelle, les tables globales DynamoDB utilisent un rapprochement de type priorité à la dernière écriture entre des mises à jour quasi-simultanées, où DynamoDB s'efforce de déterminer la dernière écriture. Avec ce mécanisme de résolution de conflits, tous les réplicas s'accordent sur la dernière mise à jour et convergent vers un état dans lequel ils ont tous des données identiques.

Disponibilité et durabilité

Si une région AWS vient à être isolée ou dégradée, votre application peut rediriger vers une autre région et effectuer des lectures et écritures dans une table de réplique différente. Vous pouvez appliquer une logique métier personnalisée pour déterminer quand rediriger des demandes vers d'autres régions.

Si une région vient à être isolée ou dégradée, DynamoDB conserve une trace des écritures effectuées qui n'ont pas encore été propagées à toutes les tables de réplique. Lorsque la région revient en ligne, DynamoDB reprend la propagation des écritures en attente de cette région vers les tables de réplique dans les autres régions. Il reprend également la propagation des écritures des autres tables de réplique vers la région revenue en ligne. Tous les écrits précédemment réussis finiront par être diffusés, quelle que soit la durée de l'isolement de la région.

Bonnes pratiques et exigences pour la gestion des tables globales

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
> Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Les tables globales Amazon DynamoDB vous permettent de répliquer vos données de table dans plusieurs régions AWS. Pour garantir une réplication correcte des données, il est important que les tables de réplique et les index secondaires dans votre table globale aient des paramètres de capacité d'écriture identiques.

Rubriques

- Version des tables globales
- Configuration requise pour l'ajout d'une nouvelle table de réplica
- Bonnes pratiques et exigences pour la gestion de la capacité

Version des tables globales

Deux versions des tables globales DynamoDB sont disponibles : Tables globales version 2019.11.21 (actuelle) et Tables globales de la version 2017.11.29 (héritée). Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).

Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales existantes de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Mise à niveau des tables globales.

Configuration requise pour l'ajout d'une nouvelle table de réplica

Si vous souhaitez ajouter une nouvelle table de réplique à une table globale, chacune des conditions suivantes doit être vrais :

- La table doit avoir la même clé de partition que tous les réplicas.
- La table doit avoir les mêmes paramètres de gestion de la capacité d'écriture que tous les réplicas.
- La table doit avoir la même clé de partition que tous les réplicas.
- DynamoDB Streams doit être activé pour la table, et le flux doit contenir à la fois la nouvelle image et l'ancienne image de l'élément.
- Aucune des tables de réplique nouvelles ou existantes dans la table globale ne peut contenir de données.

Si des index secondaires globaux sont spécifiés, les conditions suivantes doivent également être remplies :

- Les index secondaires globaux doivent avoir le même nom.
- Les index secondaires globaux doivent avoir la même clé de partition et la même clé de tri (le cas échéant).

> **Important**
>
> Les paramètres de capacité d'écriture doivent être définis de manière cohérente sur la totalité des tables de réplique de vos tables globales et des index secondaires correspondants. Pour mettre à jour les paramètres de capacité d'écriture de votre table globale, nous vous recommandons vivement d'utiliser la console DynamoDB ou l'opération d'API `UpdateGlobalTableSettings`. L'opération `UpdateGlobalTableSettings` applique automatiquement les modifications apportées aux paramètres de capacité d'écriture à l'ensemble des tables de réplique et des index secondaires correspondants dans une table globale. Si vous utilisez les opérations `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, vous devez appliquer la modification individuellement à chaque table de réplique et à l'index secondaire correspondant. Pour plus d'informations, consultez UpdateGlobalTableSettings dans la Référence d'API Amazon DynamoDB.
> Nous vous recommandons vivement d'activer la scalabilité automatique pour gérer les paramètres de capacité d'écriture approvisionnée. Si vous préférez gérer manuellement les paramètres de capacité d'écriture, vous devez approvisionner toutes vos tables de réplique en unités de capacité d'écriture répliquée égales. Approvisionnez également en unités de capacité d'écriture répliquée égales les index secondaires correspondants dans votre table globale.
> Vous devez aussi disposer d'autorisations Gestion des identités et des accès AWS (IAM) appropriées. Pour plus d'informations, consultez Utilisation d'IAM avec des tables globales.

Bonnes pratiques et exigences pour la gestion de la capacité

Lors de la gestion des paramètres de capacité pour des tables de réplique dans DynamoDB, tenez compte de ce qui suit.

Utilisation de la scalabilité automatique de DynamoDB

La scalabilité automatique de DynamoDB est la méthode recommandée pour gérer les paramètres de capacité de débit pour les tables de réplique utilisant le mode approvisionné. La scalabilité automatique de DynamoDB ajuste automatiquement les unités de capacité de lecture (RCU) et les

unités de capacité d'écriture (WCU) pour chaque table de réplique en fonction de la charge de travail réelle de votre application. Pour plus d'informations, consultez Gestion automatique de la capacité de débit avec la scalabilité automatique de DynamoDB.

Si vous créez vos tables de réplique à l'aide de l'AWS Management Console, la scalabilité automatique est activée par défaut pour chaque table de réplique, avec des paramètres de scalabilité automatique par défaut pour la gestion des unités de capacité de lecture et d'écriture.

Les modifications apportées aux paramètres de scalabilité automatique d'une table de réplique ou d'un index secondaire via la console DynamoDB ou à l'aide de l'appel `UpdateGlobalTableSettings` sont appliquées automatiquement à l'ensemble des tables de réplique et des index secondaires correspondants dans la table globale. Ces modifications remplacent tous les paramètres de scalabilité automatique existants. Cela garantit que les paramètres de capacité d'écriture approvisionnée sont cohérents entre les tables de réplique et les index secondaires au sein de votre table globale. Si vous utilisez les appels `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, vous devez appliquer la modification individuellement à chaque table de réplique et à l'index secondaire correspondant.

> **Note**
>
> Si la scalabilité automatique ne répond pas aux changements de capacité de votre application (charge de travail imprévisible), ou si vous ne souhaitez pas configurer ses paramètres (paramètres cibles pour les seuils minimum, maximum ou d'utilisation), vous pouvez utiliser le mode à la demande pour gérer la capacité de vos tables globales. Pour plus d'informations, consultez Mode de capacité à la demande.
>
> Si vous activez le mode à la demande sur une table globale, votre consommation d'unités de demande d'écriture répliquée sera cohérente avec la manière dont celles-ci sont approvisionnées. Par exemple, si vous effectuez 10 écritures dans une table locale répliquée dans deux régions supplémentaires, vous consommez 60 unités de demande d'écriture (10 + 10 = 30 et 30 x 2 = 60). Les 60 unités de demande d'écriture consommées incluent l'écriture supplémentaire consommée par les tables globales Version 2017.11.29 (héritée) pour mettre à jour les attributs `aws:rep:deleting`, `aws:rep:updatetime` et `aws:rep:updateregion`.

Gestion manuelle de la capacité

Si vous décidez de ne pas utiliser la scalabilité automatique de DynamoDB, vous devez définir manuellement les paramètres de capacité de lecture et d'écriture sur chaque table de réplique et index secondaire.

Les unités de capacité d'écriture répliquée approvisionnées sur chaque table de réplique doivent être définies sur deux fois le nombre total d'unités nécessaires pour les écritures d'application dans toutes les régions. Cela permet d'accueillir les écritures d'application qui se produisent dans la région locale et les écritures d'application répliquées provenant d'autres régions. Par exemple, supposons que vous attendez 5 écritures par seconde dans votre table de réplique en Ohio, et 5 écritures par seconde dans votre table de réplique en Virginie du Nord. Dans ce cas, vous devez approvisionner 20 unités rWCU pour chaque table de réplique (5 + 5 = 10 et 10 x 2 = 20).

Pour mettre à jour les paramètres de capacité d'écriture de votre table globale, nous vous recommandons vivement d'utiliser la console DynamoDB ou l'opération d'API `UpdateGlobalTableSettings`. L'opération `UpdateGlobalTableSettings` applique automatiquement les modifications apportées aux paramètres de capacité d'écriture à l'ensemble des tables de réplique et des index secondaires correspondants dans une table globale. Si vous utilisez les opérations `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, vous devez appliquer la modification individuellement à chaque table de réplique et à l'index secondaire correspondant. Pour plus d'informations, consultez la Référence d'API Amazon DynamoDB.

> **Note**
>
> Pour mettre à jour les paramètres (`UpdateGlobalTableSettings`) pour une table globale dans DynamoDB, vous devez disposer des autorisations `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` et `application-autoscaling:DeregisterScalableTarget`. Pour plus d'informations, consultez Utilisation d'IAM avec des tables globales.

Création d'une table globale

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la

version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Cette section explique comment créer une table globale à l'aide de la console Amazon DynamoDB ou de l'AWS Command Line Interface (AWS CLI).

Rubriques

- Création d'une table globale (console)
- Création d'une table globale (AWS CLI)

Création d'une table globale (console)

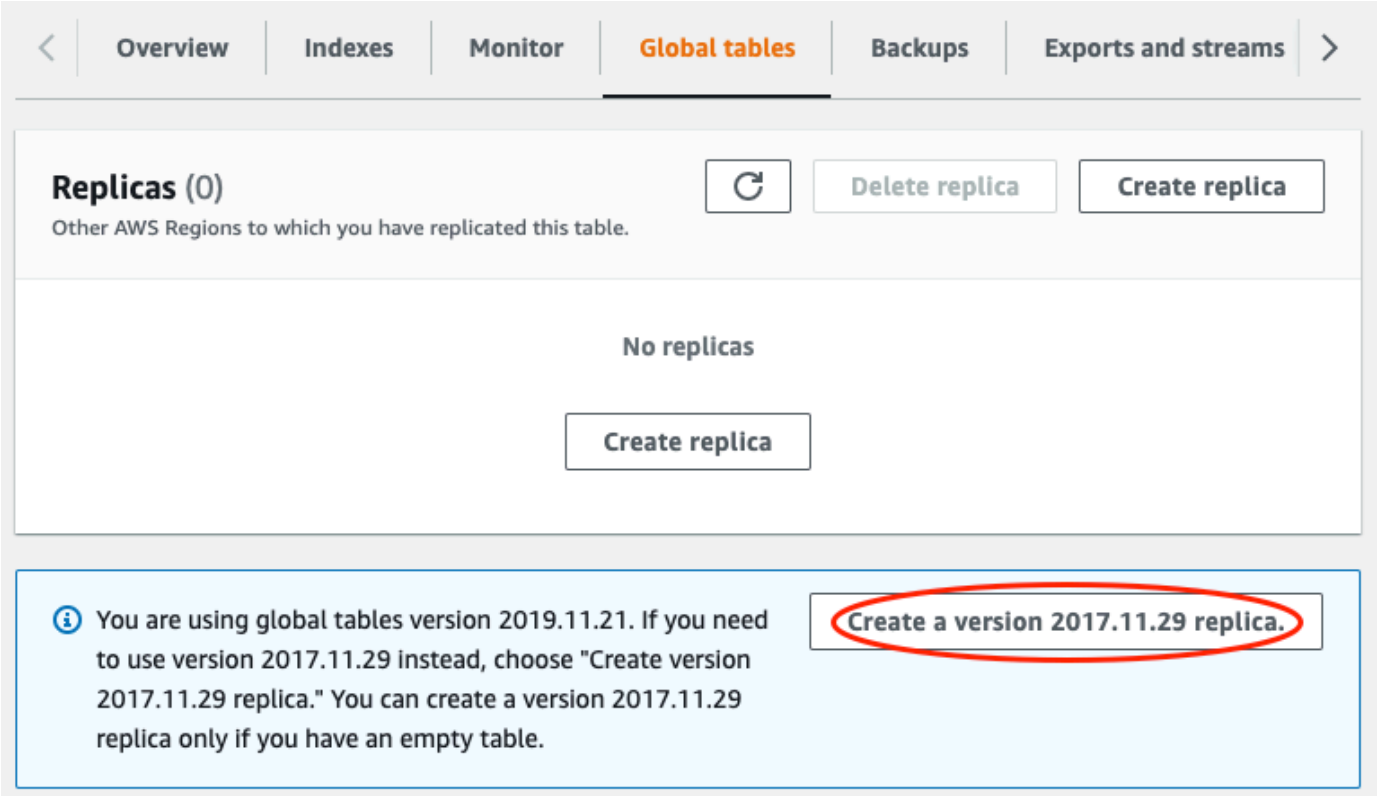
Suivez ces étapes pour créer une table globale à l'aide de la console. L'exemple suivant crée une table globale avec des tables de réplica aux États-Unis et en Europe.

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/home. Pour cet exemple, choisissez la région us-east-2 (USA Est (Ohio)).
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Tables.
3. Choisissez Créer une table.

   Sous Nom du tableau, saisissez **`Music`**.

   Pour Primary key (Clé primaire) entrez **`Artist`**. Choisissez Add sort key (Ajouter une clé de tri), puis entrez **`SongTitle`**. (**`Artist`** et **`SongTitle`** doivent tous deux être des chaînes.)

   Pour créer la table, choisissez Create. La table sert de première table de réplica dans une nouvelle table globale. Il s'agit du prototype pour d'autres tables de réplica que vous ajouterez ultérieurement.
4. Choisissez l'onglet Tables globales, puis choisissez Créer un réplica version 2017.11.29 (héritée).



5. De la liste déroulante Régions de réplication disponibles, choisissez USA Ouest (Oregon).

   La console vérifie qu'il n'existe pas de table du même nom dans la région sélectionnée. (Si une table du même nom existe, vous devez la supprimer avant de pouvoir créer une nouvelle table de réplicas dans cette région.)

6. Choisissez Créer un réplica. Cela a pour effet de démarrer le processus de création de table dans la région USA Ouest (Oregon).

   L'onglet Table globale pour la table sélectionnée (et pour toutes les autres tables de réplica) indique que la table a été répliquée dans plusieurs régions.

7. Vous allez maintenant ajouter une autre région, de sorte que votre table globale soit répliquée et synchronisée à travers les États-Unis et l'Europe. Pour ce faire, répétez l'étape 5, mais cette fois en spécifiant Europe (Francfort) au lieu de USA Ouest (Oregon).

8. Vous devez toujours utiliser AWS Management Console dans la région USA Est (Ohio). Sélectionnez Eléments dans le menu de navigation de gauche, sélectionnez le table Musique, puis choisissez Créer un élément.

   a. Pour Artist (Artiste), saisissez **`item_1`**.

   b. Pour SongTitle (Titre de chanson), saisissez **`Song Value 1`**.

c. Pour écrire l'élément, choisissez Créer un élément.

9. Après quelques instants, l'élément est répliqué dans les trois régions de votre table globale. Pour vérifier cela, sur la console, dans le sélecteur de région situé en haut à droite, choisissez Europe (Francfort). La table `Music` dans la région Europe (Francfort) doit contenir le nouvel élément.
10. Répétez l'étape 9 et choisissez USA Ouest (Oregon) pour vérifier la réplication dans cette région.

Création d'une table globale (AWS CLI)

Suivez ces étapes pour créer une table globale `Music` avec AWS CLI. L'exemple suivant crée une table globale avec des tables de réplica aux États-Unis et en Europe.

1. Créez une table (`Music`) dans la région USA Est (Ohio), avec DynamoDB Streams activé (`NEW_AND_OLD_IMAGES`).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. Créez une table `Music` identique dans la région USA Est (Virginie du Nord).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
```

```
    --region us-east-1
```

3. Créez une table globale (`Music`) composée de tables de réplique dans les régions `us-east-2` et `us-east-1`.

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> Le nom de la table globale (`Music`) doit correspondre au nom de chacune des tables de réplique (`Music`). Pour plus d'informations, consultez Bonnes pratiques relatives aux tables globales.

4. Créez une autre table en Europe (Irlande), avec les mêmes paramètres que ceux que vous avez créés aux étape 1 et 2.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

Une fois cette étape accomplie, ajoutez la nouvelle table à la table globale `Music`.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. Pour vérifier que la réplication fonctionne, ajoutez un élément à la table `Music` dans la région USA Est (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Patientez quelques secondes, puis vérifiez que l'élément a été correctement répliqué dans les régions USA Est (Virginie du Nord) et Europe (Irlande).

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Surveillance des tables globales

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
> Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (héritée) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Vous pouvez utiliser Amazon CloudWatch pour contrôler le comportement et les performances d'une table globale. Amazon DynamoDB publie les métriques `ReplicationLatency` et `PendingReplicationCount` pour chaque réplica dans la table globale.

- **`ReplicationLatency`** – Temps écoulé entre le moment où un élément mis à jour apparaît dans le flux DynamoDB pour une table de réplique, et le moment où il apparaît dans un autre réplica dans la table globale. La valeur `ReplicationLatency` est exprimée en millisecondes et émise pour chaque paire région source/région de destination.

  En mode de fonctionnement normal, la valeur de `ReplicationLatency` doit être relativement constante. Une valeur élevée pour `ReplicationLatency` peut indiquer que les mises à jour d'un réplica ne sont pas propagées vers d'autres tables de réplica dans un délai raisonnable. Avec le temps, les autres tables de réplica risquent de prendre du retard du fait qu'elles ne reçoivent plus les mises à jour de manière cohérente. Dans ce cas, vérifiez que les unités de capacité de lecture (RCU) et les unités de capacité d'écriture (WCU) sont identiques pour chaque table de réplique. De plus, lorsque vous choisissez les paramètres d'unités de capacité d'écriture, vous devez suivre les recommandations indiquées dans Version des tables globales.

  La valeur de `ReplicationLatency` peut augmenter si une région AWS se dégrade et si vous avez une table de réplique dans cette région. Dans ce cas, vous pouvez rediriger temporairement les activités de lecture et d'écriture de votre application dans une autre région AWS.

- **`PendingReplicationCount`** – Nombre de mises à jour d'élément qui sont écrites dans une table de réplique, mais ne le sont pas encore dans un autre réplica de la table globale. La valeur de `PendingReplicationCount` est exprimée en nombre d'éléments et émise pour chaque paire région source/région de destination.

  Dans le cadre d'une opération normales, la valeur de `PendingReplicationCount` devrait être très basse. Si la valeur de `PendingReplicationCount` augmente pendant des périodes prolongées, vérifiez si les paramètres de capacité d'écriture approvisionnée de vos tables de réplique sont appropriées pour votre charge de travail actuelle.

  La valeur de `PendingReplicationCount` peut augmenter si une région AWS se dégrade et si vous avez une table de réplique dans cette région. Dans ce cas, vous pouvez rediriger temporairement les activités de lecture et d'écriture de votre application dans une autre région AWS.

Pour plus d'informations, consultez Métriques et dimensions DynamoDB.

Utilisation d'IAM avec des tables globales

> **Important**
>
> Cette documentation concerne la version 2017.11.29 (héritée) des tables globales, qui doit être évitée pour les nouvelles tables globales. Les clients doivent utiliser la version 2019.11.21 (actuelle) des tables globales dans la mesure du possible, car elle offre une plus grande flexibilité, une efficacité accrue et utilise moins de capacité d'écriture que la version 2017.11.29 (héritée).
> Pour déterminer quelle version vous utilisez, consultez Détermination de la version d'une table globale. Pour mettre à jour les tables globales de la version 2017.11.29 (ancienne) vers la version 2019.11.21 (actuelle), consultez Versions des tables globales DynamoDB.

Lorsque vous créez une table globale pour la première fois, Amazon DynamoDB crée automatiquement pour vous un rôle lié à un service Gestion des identités et des accès AWS(IAM). Ce rôle nommé permet à DynamoDB de gérer pour vous la réplication entre régions pour les tables globales. Ne supprimez pas ce rôle lié à un service. Dans ce cas, la totalité de vos tables globales cessent de fonctionner.

Pour en savoir plus sur l'utilisation des rôles liés à un service, consultez Utilisation des rôles liés à un service dans le Guide de l'utilisateur IAM.

Pour créer et gérer des tables globales dans DynamoDB, vous devez disposer de l'autorisation `dynamodb:CreateGlobalTable` pour accéder à chacun des éléments suivants :

- La table de réplicas que vous souhaitez ajouter.
- Chaque réplica existant faisant déjà partie de la table globale.
- La table globale proprement dite.

Pour mettre à jour les paramètres (`UpdateGlobalTableSettings`) pour une table globale dans DynamoDB, vous devez disposer des autorisations `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` et `application-autoscaling:DeregisterScalableTarget`.

Les autorisations `application-autoscaling:DeleteScalingPolicy` et `application-autoscaling:DeregisterScalableTarget` sont requises lors de la mise à jour d'une politique

de mise à l'échelle existante. Le service de tables globales peut ainsi supprimer l'ancienne politique de mise à l'échelle avant d'attacher la nouvelle politique à la table ou à l'index secondaire.

Si vous utilisez une politique IAM pour gérer l'accès à une table de réplique, vous devez appliquer une politique identique à tous les autres réplicas au sein de cette table globale. Cette pratique vous aide à maintenir un modèle d'autorisations cohérent pour toutes les tables de réplique.

En utilisant des politiques IAM identiques sur tous les réplicas d'une table globale, vous pouvez également éviter d'accorder un accès involontaire de lecture et d'écriture aux données de votre table globale. Par exemple, considérons un utilisateur qui n'a accès qu'à un seul réplica dans une table globale. Si cet utilisateur peut écrire dans ce réplica, DynamoDB propage l'écriture à toutes les autres tables de réplique. En effet, l'utilisateur peut écrire (indirectement) dans tous les autres réplicas au sein de la table globale. Vous pouvez éviter ce scénario en utilisant des politiques IAM cohérentes sur toutes les tables de réplique.

Exemple : autoriser l' CreateGlobalTableaction

Afin de pouvoir ajouter un réplica à une table globale, vous devez disposer de l'autorisation `dynamodb:CreateGlobalTable` pour la table globale et pour chacune de ses tables de réplique.

La politique IAM suivante autorise l'action `CreateGlobalTable` sur toutes les tables.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

Exemple : Autoriser les actions UpdateGlobalTable, DescribeLimits, application-autoscaling : et application-autoscaling : DeleteScalingPolicy DeregisterScalableTarget

Pour mettre à jour les paramètres (`UpdateGlobalTableSettings`) pour une table globale dans DynamoDB, vous devez disposer des autorisations `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` et `application-autoscaling:DeregisterScalableTarget`.

La politique IAM suivante autorise l'action `UpdateGlobalTableSettings` sur toutes les tables.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Exemple : autoriser l' CreateGlobalTable action pour un nom de table global spécifique avec des répliques autorisées dans certaines régions uniquement

La politique IAM suivante accorde des autorisations pour autoriser l'action `CreateGlobalTable` permettant de créer une table globale nommée `Customers` avec des réplicas dans deux régions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```

Version précédente de l'API DynamoDB de bas niveau (2011-12-05)

Cette section documente les opérations disponibles dans la version précédente de l'API de bas niveau DynamoDB (2011-12-05). Cette version de l'API de bas niveau est conservée à des fins de compatibilité descendante avec des applications existantes.

Les nouvelles applications doivent utiliser la version actuelle de l'API (2012-08-10). Pour de plus amples informations, veuillez consulter Référence de l'API DynamoDB.

> **Note**
>
> Nous vous recommandons de migrer vos applications vers la dernière version de l'API (2012-08-10), car les nouvelles fonctions de DynamoDB ne seront pas rétroportées vers la version précédente de l'API.

Rubriques

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables

- GetItem
- ListTables
- PutItem
- Requête
- Analyser
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

L'opération `BatchGetItem` renvoie les attributs de plusieurs éléments de tables multiples en utilisant leurs clés primaires. Le nombre maximum d'éléments pouvant être extraits pour une seule opération est de 100. En outre, le nombre d'éléments récupérés est limité par une limite de taille de 1 Mo. Si la limite de taille de réponse est dépassée ou si un résultat partiel est renvoyé parce que le débit approvisionné de la table est dépassé ou en raison d'un échec de traitement interne, DynamoDB renvoie une valeur `UnprocessedKeys` afin de vous permettre de réessayer l'opération en commençant par l'élément suivant à obtenir. DynamoDB ajuste automatiquement le nombre d'éléments renvoyés par page pour appliquer cette limite. Par exemple, même si vous demandez à récupérer 100 éléments ayant chacun une taille de 50 Ko, le système renvoie 20 éléments et une valeur `UnprocessedKeys` appropriée pour vous permettre d'obtenir la page de résultats suivante. Si vous le souhaitez, votre application peut inclure sa propre logique pour assembler les pages de résultats en un seul ensemble.

Si aucun élément n'a pu être traité en raison d'un débit approvisionné insuffisant sur chacune des tables impliquées dans la demande, DynamoDB renvoie une erreur `ProvisionedThroughputExceededException`.

> **Note**
>
> Par défaut, la commande `BatchGetItem` effectue des lectures éventuellement cohérentes sur chaque table dans la demande. Vous pouvez définir le paramètre `ConsistentRead` sur `true` pour chaque table si vous voulez plutôt des lectures cohérentes.
> L'opération `BatchGetItem` récupère des éléments en parallèle afin de réduire les latences de réponse.
> Lors de la conception de votre application, gardez à l'esprit que DynamoDB ne garantit pas l'ordre des attributs dans la réponse renvoyée. Incluez les valeurs de clé primaire dans le paramètre `AttributesToGet` pour les éléments dans votre demande afin de vous aider à analyser la réponse par élément.
> Si les éléments demandés n'existent pas, rien n'est renvoyé dans la réponse pour ces éléments. Les demandes d'éléments inexistants consomment les unités de capacité de lecture minimum en fonction du type de lecture. Pour de plus amples informations, consultez Tailles et formats d'élément DynamoDB.

Requêtes

Syntaxe

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
```

```
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

Name (Nom)	Description	Obligatoire
`RequestItems`	Conteneur du nom de table et éléments correspondants à obtenir par la clé primaire. Lors de la demande d'élément s, chaque nom de table ne peut être appelé qu'une seule fois par opération. Type : String Par défaut : aucun	Oui
`Table`	Nom de la table contenant les éléments à obtenir. L'entrée est simplement une chaîne spécifiant une table existante sans libellé. Type : String Par défaut : aucun	Oui
`Table:Keys`	Valeurs de clé primaire définissant les éléments dans la table spécifiée. Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : clés	Oui

Name (Nom)	Description	Obligatoire
`Table:AttributesToGet`	Tableau de noms d'attribut dans la table spécifiée. Si des noms d'attribut ne sont pas spécifiés, tous les attributs sont renvoyés. Si certains attributs ne sont pas trouvés, ils n'apparaissent pas dans le résultat. Type : Array	Non
`Table:ConsistentRead`	Si la valeur est définie sur `true`, une lecture cohérente est effectuée. Sinon une cohérence éventuelle est utilisée. Type : booléen	Non

Réponses

Syntaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
```

```
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
}
```

Name (Nom)	Description
`Responses`	Noms de table et attributs d'élément respectifs des tables. Type : carte
`Table`	Nom de la table contenant les éléments. L'entrée est simplement une chaîne spécifiant la table sans libellé. Type : String

Name (Nom)	Description
`Items`	Conteneur pour les noms d'attribut et les valeurs correspondant aux paramètres d'opération. Type : mappage de noms d'attribut à leurs types de données et valeurs.
`ConsumedCapacityUnits`	Nombre d'unités de capacité de lecture consommées pour chaque table. Cette valeur indique le nombre appliqué à votre débit approvisionné. Les demandes d'éléments inexistants consomment les unités de capacité de lecture minimum, selon le type de lecture. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre
`UnprocessedKeys`	Contient un tableau de tables et leurs clés respectives qui n'ont pas été traités avec la réponse actuelle, probablement en raison de l'atteinte d'une limite de taille de réponse. La valeur `UnprocessedKeys` se présente sous la même forme qu'un paramètre `RequestItems` (de sorte que la valeur peut être fournie directement à une opération `BatchGetItem` subséquente). Pour plus d'informations, consultez le paramètre `RequestItems` ci-dessus. Type : Array

Name (Nom)	Description
`UnprocessedKeys` : `Table`: `Keys`	Valeurs d'attribut de clé primaire qui définissent les éléments et les attributs associés aux éléments. Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : tableau de paires nom-valeur d'attribut.
`UnprocessedKeys` : `Table`: `AttributesToGet`	Noms d'attribut dans la table spécifiée. Si des noms d'attribut ne sont pas spécifiés, tous les attributs sont renvoyés. Si certains attributs ne sont pas trouvés, ils n'apparaissent pas dans le résultat. Type : tableau de noms d'attribut.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Si la valeur est définie sur `true`, une lecture cohérente est utilisée pour la table spécifiée. Sinon, une lecture éventuellement cohérente est utilisée. Type : booléen.

Erreurs spéciales

Erreur	Description
`ProvisionedThroughputExceededException`	Votre débit approvisionné autorisé maximum a été dépassé.

Exemples

Les exemples suivants illustrent une requête HTTP POST et une réponse utilisant l'opération BatchGetItem. Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation d’éléments et d’attributs dans DynamoDB.

Exemple de demande

L'exemple suivant demande des attributs de deux tables différentes.

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

Exemple de réponse

L'exemple suivant est la réponse.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
```

```
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":{}
}
```

BatchWriteItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Cette opération vous permet d'insérer et de supprimer plusieurs éléments dans plusieurs tables en un seul appel.

Pour charger un élément, vous pouvez utiliser `PutItem`. Et pour supprimer un élément, vous pouvez utiliser `DeleteItem`. Toutefois, lorsque vous souhaitez charger ou supprimer de grandes quantités de données, par exemple en chargeant des données d'Amazon EMR (Amazon EMR) ou en migrant des données d'une autre base de données vers DynamoDB, `BatchWriteItem` offre une alternative efficace.

Si vous utilisez des langages tels que Java, vous pouvez utiliser des unités d'exécution pour charger des éléments en parallèle. Cela rend votre application plus complexe en lien avec la gestion des unités d'exécution. D'autres langages ne prennent pas en charge les unités d'exécution. Par exemple, si vous utilisez PHP, vous devez charger ou supprimer les éléments un par un. Dans les deux cas, `BatchWriteItem` fournit une alternative où les actions d'insertion et de suppression d'éléments spécifiées sont traitées en parallèle. Vous bénéficiez ainsi de la puissance de l'approche de groupe d'unités d'exécution sans avoir à introduire de la complexité dans votre application.

Notez que chaque action d'insertion ou de suppression spécifiée dans une opération `BatchWriteItem` a le même coût en termes d'unités de capacité consommées. Cependant, `BatchWriteItem` effectuant les actions spécifiées en parallèle, vous bénéficiez d'une latence plus

faible. Les actions de suppression d'éléments inexistants consomment 1 unité de capacité d'écriture. Pour plus d'informations sur les unités de capacité consommées, consultez Utilisation de tables et de données dans DynamoDB.

Lorsque vous utilisez `BatchWriteItem`, notez les limitations suivantes :

- Nombre maximum d'actions dans une seule demande – Vous pouvez spécifier jusqu'à 25 actions d'insertion ou de suppression. Toutefois, la taille totale de la demande ne peut pas dépasser 1 Mo (charge utile HTTP).
- Vous pouvez utiliser l'opération `BatchWriteItem` uniquement pour insérer et supprimer des éléments. Vous ne pouvez pas l'utiliser pour mettre à jour des éléments existants.
- Opération non atomique – Les actions individuelles spécifiées dans une opération `BatchWriteItem` sont atomiques. Cependant, une opération `BatchWriteItem` en tant que tout, est une opération effectuée sur la base du meilleur effort, non une opération atomique. Autrement dit, dans une demande `BatchWriteItem`, certaines opérations peuvent réussir et d'autres échouer. Les opérations ayant échoué sont renvoyées dans un champ `UnprocessedItems` dans la réponse. Certains de ces échecs peuvent résulter d'un dépassement du débit approvisionné configuré pour la table, ou d'un échec temporaire tel qu'une erreur réseau. Vous pouvez examiner et éventuellement renvoyer les demandes. En règle générale, vous appelez l'opération `BatchWriteItem` dans une boucle et, dans chaque itération, vérifiez les éléments non traités, puis soumettez une nouvelle demande `BatchWriteItem` avec ceux-ci.
- Ne renvoie aucun élément – L'opération `BatchWriteItem` est conçue pour charger efficacement de grandes quantités de données. Il ne présente pas le même niveau de sophistication que les opérations `PutItem` et `DeleteItem`. Par exemple, l'opération `DeleteItem` prend en charge le champ `ReturnValues` dans le corps de votre demande pour demander l'élément supprimé dans la réponse. L'opération `BatchWriteItem` ne renvoie aucun élément dans la réponse.
- Contrairement aux opérations `PutItem` et `DeleteItem`, l'opération `BatchWriteItem` ne vous permet pas de spécifier des conditions sur des demandes d'écriture individuelles dans l'opération.
- Les valeurs d'attribut ne peuvent pas être nulles, les attributs de type chaîne et binaire doivent avoir une longueur supérieure à zéro, et les attributs de type ensemble ne peuvent pas être vides. Les demandes comprenant des valeurs vides sont rejetées avec une valeur `ValidationException`.

DynamoDB rejette toute l'opération d'écriture par lot si l'une des conditions suivantes est vraie :

- Si une ou plusieurs tables spécifiées dans la demande `BatchWriteItem` n'existent pas.

- Si les attributs de clé primaire spécifiés sur un élément dans la demande ne correspondent pas au schéma de clé primaire de la table correspondante.
- Si vous essayez d'effectuer plusieurs opérations sur le même élément dans la même demande `BatchWriteItem`. Par exemple, vous ne pouvez pas insérer et supprimer le même élément dans la même demande `BatchWriteItem`.
- Si la taille totale de la demande dépasse la limite de 1 Mo (charge utile HTTP).
- Si un élément individuel dans un lot dépasse la limite de taille d'élément de 64 Ko.

Requêtes

Syntaxe

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
```

```
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}
```

```
NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

Dans le corps de la demande, l'objet JSON `RequestItems` décrit les opérations que vous souhaitez effectuer. Les opérations sont regroupées par tables. Vous pouvez utiliser une opération `BatchWriteItem` pour mettre à jour ou supprimer plusieurs éléments dans plusieurs tables. Pour chaque demande d'écriture, vous devez identifier le type de demande (`PutItem`, `DeleteItem`), suivi d'informations détaillées sur l'opération.

- Pour une opération `PutRequest`, vous fournissez l'élément, c'est-à-dire une liste d'attributs avec leurs valeurs.
- Pour une opération `DeleteRequest`, vous indiquez le nom et la valeur de la clé primaire.

Réponses

Syntaxe

Voici la syntaxe du corps JSON renvoyé dans la réponse.

```
{
  "Responses" :         ConsumedCapacityUnitsByTable
  "UnprocessedItems" :  RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

Erreurs spéciales

Aucune erreur spécifique à cette opération.

Exemples

L'exemple suivant illustre une requête HTTP POST et la réponse d'une opération `BatchWriteItem`. La requête spécifie les opérations suivantes sur les tables Reply et Thread :

- Insérer et supprimer un élément dans la table Reply
- Insérer un élément dans la table Thread

Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation d'éléments et d'attributs dans DynamoDB.

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
```

```
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

Exemple de réponse

L'exemple de réponse suivant montre une opération d'insertion sur les tables Thread et Reply réussie, et une opération de suppression sur la table Reply ayant échoué (pour une raison telle qu'une limitation résultant du dépassement du débit approvisionné sur la table). Dans la réponse JSON, notez ce qui suit :

- L'objet `Responses` montre qu'une unité de capacité a été consommée sur les deux tables `Thread` et `Reply` suite à l'opération d'insertion réussie sur chacune d'elles.
- L'objet `UnprocessedItems` montre l'opération de suppression ayant échoué sur la table `Reply`. Vous pouvez ensuite émettre un nouvel appel `BatchWriteItem` pour traiter ces demandes non traitées.

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT

{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
   },
   "UnprocessedItems":{
      "Reply":[
         {
            "DeleteRequest":{
               "Key":{
                  "HashKeyElement":{
                     "S":"DynamoDB#DynamoDB Thread 4"
                  },
                  "RangeKeyElement":{
                     "S":"oops - accidental row"
                  }
               }
            }
         }
      ]
   }
}
```

CreateTable

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

L'opération `CreateTable` ajoute une table à votre compte

Le nom de la table doit être unique parmi ceux associés au compte AWS émettant la demande, et à la région AWS recevant la demande (par exemple dynamodb.us-west-2.amazonaws.com). Chaque point de terminaison DynamoDB est entièrement indépendant. Par exemple, si vous avez deux tables nommées « MyTable », l'une dans la région dynamodb.us-west-2.amazonaws.com et l'autre dans la région dynamodb.us-west-1.amazonaws.com, ces tables sont complètement indépendantes et ne partagent aucune donnée.

L'opération `CreateTable` déclenche un flux asynchrone pour commencer à créer la table. DynamoDB renvoie immédiatement l'état de la table (`CREATING`) jusqu'à ce que la table soit dans l'état `ACTIVE`. Une fois la table dans l'état `ACTIVE`, vous pouvez effectuer des opérations de plan de données.

Utilisez l'opération DescribeTables pour vérifier l'état de la table.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table à créer.	Oui

Name (Nom)	Description	Obligatoire
	Les caractères autorisés sont a-z, A-Z, 0-9, « _ » (trait de soulignement), « - » (tiret) et « . » (point). Les noms peuvent comporter entre 3 et 255 caractères. Type : String	

Name (Nom)	Description	Obligatoire
`KeySchema`	Structure (simple ou composite) de la clé primaire pour la table. Une paire nom-valeur pour l'élément `HashKeyElement` est obligatoire, et une paire nom-valeur pour l'élément `RangeKeyElement` est facultatif (obligatoire uniquement pour les clés primaires composites). Pour plus d'informations sur les clés primaires, consultez Clé primaire. Les noms d'élément de clé primaire peuvent comporter entre 1 et 255 caractères sans restriction quant au type de caractère. Les valeurs possibles pour AttributeType sont « S » (chaîne), « N » (numérique) ou « B » (binaire). Type : mappage de `HashKeyElement` , ou de `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.	Oui

Name (Nom)	Description	Obligatoire
`ProvisionedThroughput`	Nouveau débit pour la table spécifiée, composé de valeurs pour `ReadCapacityUnits` et `WriteCapacityUnits` . Pour en savoir plus, consultez Mode de capacité provisionnée DynamoDB. **Note** Pour connaître les valeurs maximum/minimum actuelles, consultez Quotas dans Amazon DynamoDB. Type : Array	Oui

Name (Nom)	Description	Obligatoire
`ProvisionedThroughput : ReadCapacityUnits`	Définit le nombre minimum de `ReadCapacityUnits` cohérentes consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations. Des opérations de lecture éventuellement cohérente nécessitant moins d'effort qu'une opération de lecture cohérente, un paramètre de 50 `ReadCapacityUnits` par seconde fournit 100 `ReadCapacityUnits` éventuellement cohérentes par seconde. Type : nombre	Oui
`ProvisionedThroughput : WriteCapacityUnits`	Définit le nombre minimum de `WriteCapacityUnits` consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations. Type : nombre	Oui

Réponses

Syntaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

Name (Nom)	Description
`TableDescription`	Conteneur pour les propriétés de la table.
`CreationDateTime`	Date à laquelle la table a été créée au format d'heure UNIX. Type : nombre
`KeySchema`	Structure (simple ou composite) de la clé primaire pour la table. Une paire nom-valeur pour l'élément `HashKeyElement` est obligatoire, et une paire nom-valeur pour l'élément `RangeKeyElement` est facultatif (obligatoire uniquement pour les clés primaires composites). Pour plus d'informations sur les clés primaires, consultez Clé primaire.

Name (Nom)	Description
	Type : mappage de `HashKeyElement` , ou de `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.
`ProvisionedThroughput`	Débit pour la table spécifiée, composé de valeurs pour `ReadCapacityUnits` et `WriteCapacityUnits` . Consultez Mode de capacité provisionnée DynamoDB. Type : Array
`ProvisionedThroughput :ReadCapacityUnits`	Nombre minimum de `ReadCapacityUnits` consommées par seconde avant que DynamoDB équilibre la charge avec d'autres opérations Type : nombre
`ProvisionedThroughput :WriteCapacityUnits`	Nombre minimum de `ReadCapacityUnits` consommées par seconde avant que `WriteCapacityUnits` équilibre la charge avec d'autres opérations Type : nombre
`TableName`	Nom de la table créée. Type : String

Name (Nom)	Description
`TableStatus`	État actuel de la table (`CREATING`). Une fois la table dans l'état `ACTIVE`, vous pouvez y insérer des données. Utilisez l'API DescribeTables pour vérifier l'état de la table. Type : String

Erreurs spéciales

Erreur	Description
`ResourceInUseException`	Essayez de recréer une table existante.
`LimitExceededException`	Le nombre de demandes de table simultané es (nombre cumulé de tables dans l'état `CREATING`, `DELETING` ou `UPDATING`) dépasse le maximum autorisé. **Note** Pour connaître les valeurs maximum/ minimum actuelles, consultez Quotas dans Amazon DynamoDB. .

Exemples

L'exemple suivant crée une table avec une clé primaire composite contenant une chaîne et un nombre. Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation de tables et de données dans DynamoDB.

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

Actions connexes

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Supprime un élément d'une table par clé primaire Vous pouvez effectuer une opération de suppression conditionnelle qui supprime l'élément s'il existe ou s'il a une valeur d'attribut attendue.

> **Note**
>
> Si vous spécifiez `DeleteItem` sans attribut ou valeur, tous les attributs de l'élément sont supprimés.
> Si vous ne spécifiez pas de condition, l'opération `DeleteItem` est opération idempotente. Son exécution à plusieurs reprises sur le même élément ou attribut n'entraîne pas de réponse d'erreur.
> Les suppressions conditionnelles sont utiles pour ne supprimer des éléments et attributs que si des conditions spécifiques sont remplies. Si les conditions sont remplies, DynamoDB effectue la suppression. Sinon, l'élément n'est pas supprimé.
> Vous pouvez effectuer la vérification conditionnelle attendue sur un attribut par opération.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
```

```
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table contenant l'élément à supprimer. Type : String	Oui
`Key`	Clé primaire définissant l'élément. Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : mappage de `HashKeyElement` à sa valeur, et de `RangeKeyElement` à sa valeur.	Oui
`Expected`	Désigne un attribut pour une suppression conditionnelle. Le paramètre `Expected` vous permet de fournir un nom d'attribut et de spécifier si DynamoDB doit ou non vérifier si l'attribut a une valeur particulière avant de le supprimer. Type : mappage de noms d'attribut.	Non
`Expected:AttributeName`	Nom de l'attribut pour l'insertion conditionnelle.	Non

Name (Nom)	Description	Obligatoire
	Type : String	

Name (Nom)	Description	Obligatoire

`Expected:AttributeName: ExpectedAttributeValue`	Utilisez ce paramètre pour spécifier si une valeur existe déjà pour la paire nom-valeur de l'attribut.
	
	La notation JSON suivante supprime l'élément si l'attribut « Color » (Couleur) n'existe pas pour cet élément :
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La notation JSON suivante vérifie si l'attribut « Color » (Couleur) a une valeur existante de « Yellow » (Jaune) avant de supprimer l'élément :
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Par défaut, si vous utilisez le paramètre `Expected` et fournissez une `Value`, DynamoDB suppose que l'attribut existe et a une valeur actuelle à remplacer. Vous n'avez donc pas à spécifier `{"Exists":true}` , car c'est implicite. Vous pouvez raccourcir la demande pour :
		Non

Name (Nom)	Description	Obligatoire
	`"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` **Note** Si vous spécifiez `{"Exists":true}` sans valeur d'attribut à vérifier, DynamoDB renvoie une erreur.	
`ReturnValues`	Utilisez ce paramètre si vous souhaitez obtenir les paires nom-valeur d'attribut avant leur suppression. Les valeurs de paramètre possibles sont `NONE` (par défaut) ou `ALL_OLD`. Si `ALL_OLD` est spécifié, le contenu de l'ancien élément est renvoyé. Si ce paramètre n'est pas fourni ou est `NONE`, rien n'est retourné. Type : String	Non

Réponses

Syntaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT
```

```
{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
}
```

Name (Nom)	Description
`Attributes`	Si le paramètre `ReturnValues` est fourni en tant que `ALL_OLD` dans la demande, DynamoDB renvoie un tableau de paires nom-valeur d'attribut (essentiellement, l'élément supprimé). Dans le cas contraire, la réponse contient un ensemble vide. Type : tableau de paires nom-valeur d'attribut.
`ConsumedCapacityUnits`	Nombre d'unités de capacité d'écriture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Les demandes de suppressi on d'éléments inexistants consomment 1 unité de capacité d'écriture. Pour plus d'informa tions, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Erreur	Description
`ConditionalCheckFailedException`	Le contrôle conditionnel a échoué. La valeur d'attribut attendue n'a pas été trouvée.

Exemples

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
}
```

Actions connexes

- PutItem

DeleteTable

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

L'opération `DeleteTable` supprime une table et la totalité de ses éléments. Après une demande `DeleteTable`, la table spécifiée est dans l'état `DELETING` jusqu'à ce que DynamoDB termine la suppression. Si la table est dans l'état `ACTIVE`, vous pouvez la supprimer. Si une table est dans l'état `CREATING` ou `UPDATING`, DynamoDB renvoie un erreur `ResourceInUseException`. Si la table spécifiée n'existe pas, DynamoDB renvoie une erreur `ResourceNotFoundException`. Si la table est déjà dans l'état `DELETING`, aucune erreur n'est renvoyée.

> **Note**
>
> DynamoDB peut continuer à accepter les demandes d'opération de plan de données, telles que `GetItem` et `PutItem`, sur une table dans l'état `DELETING` jusqu'à ce que la suppression de celle-ci soit terminée.

Les table sont uniques parmi celles associées au compte AWS émettant la demande, et à la région AWS recevant la demande (par exemple dynamodb.us-west-1.amazonaws.com). Chaque point de terminaison DynamoDB est entièrement indépendant. Par exemple, si vous avez deux tables nommées « MyTable », l'une dans la région dynamodb.us-west-2.amazonaws.com et l'autre dans la région dynamodb.us-west-1.amazonaws.com, ces tables sont complètement indépendantes et ne partagent aucune donnée. La suppression de l'une d'elles n'entraîne pas la suppression de l'autre.

Utilisez l'opération DescribeTables pour vérifier l'état de la table.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table à supprimer. Type : String	Oui

Réponses

Syntaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"DELETING"
    }
}
```

Name (Nom)	Description
`TableDescription`	Conteneur pour les propriétés de la table.
`CreationDateTime`	Date de création de la table. Type : nombre
`KeySchema`	Structure (simple ou composite) de la clé primaire pour la table. Une paire nom-valeur pour l'élément `HashKeyElement` est obligatoire, et une paire nom-valeur pour l'élément `RangeKeyElement` est facultatif (obligatoire uniquement pour les clés primaires composites). Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : mappage de `HashKeyElement` , ou de `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.
`ProvisionedThroughput`	Débit pour la table spécifiée, composé de valeurs pour `ReadCapacityUnits` et `WriteCapacityUnits` . Consultez Mode de capacité provisionnée DynamoDB.
`ProvisionedThroughput` : `ReadCapacityUnits`	Nombre minimum de `ReadCapacityUnits` consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations. Type : nombre
`ProvisionedThroughput` : `WriteCapacityUnits`	Nombre minimum de `WriteCapacityUnits` consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations.

Name (Nom)	Description
	Type : nombre
`TableName`	Nom de la table supprimée. Type : String
`TableStatus`	État actuel de la table (`DELETING`). Une fois la table supprimée, les demandes suivantes pour la table renvoient `resource not found`. Utilisez l'opération DescribeTables pour vérifier l'état de la table. Type : String

Erreurs spéciales

Erreur	Description
`ResourceInUseException`	La table est dans l'état `CREATING` ou `UPDATING` et ne peut pas être supprimée.

Exemples

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

Actions connexes

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Renvoie des informations sur la table, dont son état actuel, son schéma de clé primaire et sa date de création. Les résultats de l'opération DescribeTable sont éventuellement cohérents. Si vous utilisez l'opération DescribeTable trop tôt dans le processus de création d'une table, DynamoDB renvoie une erreur `ResourceNotFoundException`. Si vous utilisez l'opération DescribeTable trop tôt dans le processus de mise à jour d'une table, il se peut que les nouvelles valeurs ne soient pas immédiatement disponibles.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table à décrire. Type : String	Oui

Réponses

Syntaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
```

```
}
```

Name (Nom)	Description
`Table`	Conteneur pour la table décrite. Type : String
`CreationDateTime`	Date à laquelle la table a été créée au format d'heure UNIX.
`ItemCount`	Nombre d'éléments de la table spécifiée. DynamoDB met à jour cette valeur environ toutes les six heures. Il se peut que cette valeur ne reflète pas des modifications récentes. Type : nombre
`KeySchema`	Structure (simple ou composite) de la clé primaire pour la table. Une paire nom-valeur pour l'élément `HashKeyElement` est obligatoire, et une paire nom-valeur pour l'élément `RangeKeyElement` est facultatif (obligatoire uniquement pour les clés primaires composites). La taille maximum de clé de hachage est de 2 048 octets. La taille maximum de clé de plage est de 1 024 octets. Les deux limites sont appliquées séparément (c'est-à-dire que vous pouvez avoir une taille de clé combinée de hachage et de plage égale à 2 048 + 1 024 octets). Pour plus d'informations sur les clés primaires, consultez Clé primaire.
`ProvisionedThroughput`	Débit de la table spécifiée, composé de valeurs pour `LastIncreaseDateTime` (le cas échéant), `LastDecreaseDateTime` (le cas échéant), `ReadCapacityUnits` et `WriteCapacityUnits`. Si le débit de la

Name (Nom)	Description
	table n'a jamais été augmenté ou diminué, DynamoDB ne renvoie pas de valeurs pour ces éléments. Consultez Mode de capacité provisionnée DynamoDB. Type : Array
`TableName`	Le nom de la table demandée. Type : String
`TableSizeBytes`	Taille totale de la table spécifiée, en octets. DynamoDB met à jour cette valeur environ toutes les six heures. Il se peut que cette valeur ne reflète pas des modifications récentes. Type : nombre
`TableStatus`	État actuel de la table (`CREATING`, `ACTIVE`, `DELETING` ou `UPDATING`). Une fois la table dans l'état `ACTIVE`, vous pouvez ajouter des données.

Erreurs spéciales

Il n'existe pas d'erreur spécifique de cette opération.

Exemples

Les exemples suivants montrent une requête et une réponse HTTP POST utilisant l'opération DescribeTable pour une table nommée « comp-table ». La table a une clé primaire composite.

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
```

```
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Exemple de réponse

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

Actions connexes

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

L'opération `GetItem` renvoie un ensemble d'`Attributes` pour un élément correspondant à la clé primaire. Si aucun élément correspondant n'est trouvé, `GetItem` ne renvoie pas de données.

L'opération `GetItem` fournit une lecture éventuellement cohérente par défaut. Si les lectures éventuellement cohérentes ne sont pas acceptables pour votre application, utilisez `ConsistentRead`. Bien que cette opération puisse prendre plus de temps qu'une lecture standard, elle renvoie toujours la dernière valeur mise à jour. Pour de plus amples informations, consultez Cohérence en lecture DynamoDB.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table contenant l'élément demandé. Type : String	Oui
`Key`	Valeurs de clé primaire définissant l'élément. Pour plus d'informations sur les	Oui

Name (Nom)	Description	Obligatoire
	clés primaires, consultez Clé primaire. Type : mappage de `HashKeyElement` à sa valeur, et de `RangeKeyElement` à sa valeur.	
`AttributesToGet`	Tableau de noms d'attribut. Si des noms d'attribut ne sont pas spécifiés, tous les attributs sont renvoyés. Si certains attributs ne sont pas trouvés, ils n'apparaissent pas dans le résultat. Type : Array	Non
`ConsistentRead`	Si la valeur est définie sur `true`, une lecture cohérente est effectuée. Sinon une cohérence éventuelle est utilisée. Type : booléen	Non

Réponses

Syntaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 144

{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
```

```
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

Name (Nom)	Description
`Item`	Contient les attributs demandés. Type : mappage de paires nom-valeur d'attribut.
`ConsumedCapacityUnits`	Nombre d'unités de capacité de lecture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Les demandes d'éléments inexistants consomment les unités de capacité de lecture minimum, selon le type de lecture. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Aucune erreur spécifique à cette opération.

Exemples

Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation d'éléments et d'attributs dans DynamoDB.

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0
```

```
{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

Exemple de réponse

Notez que la valeur de ConsumedCapacityUnits est 1 parce que le paramètre facultatif `ConsistentRead` a la valeur `true`. Si `ConsistentRead` a la valeur `false` (ou si la valeur n'est pas spécifiée) pour la même demande, la réponse est éventuellement cohérente et la valeur de ConsumedCapacityUnits est 0,5.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

ListTables

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Renvoie un tableau de toutes les tables associées au compte et au point de terminaison actuels

Chaque point de terminaison DynamoDB est entièrement indépendant. Par exemple, si vous avez deux tables nommées « MyTable », l'une dans la région dynamodb.us-west-2.amazonaws.com et l'autre dans la région dynamodb.us-east-1.amazonaws.com, ces tables sont complètement indépendantes et ne partagent aucune donnée. L'opération ListTables renvoie tous les noms de table associés au compte faisant la demande pour le point de terminaison recevant la demande.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

Par défaut, l'opération ListTables demande tous les noms de table associés au compte faisant la demande pour le point de terminaison recevant la demande.

Nom	Description	Obligatoire
`Limit`	Nombre maximum de noms de table à renvoyer. Type : entier	Non
`ExclusiveStartTableName`	Nom de la table qui démarre la liste. Si vous avez déjà exécuté une opération ListTables et reçu une valeur `LastEvaluatedTableName` dans la réponse, utilisez cette valeur ici pour continuer la liste. Type : String	Non

Réponses

Syntaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

Name (Nom)	Description
`TableNames`	Noms des tables associées au compte actuel au point de terminaison actuel. Type : Array
`LastEvaluatedTableName`	Nom de la dernière table dans la liste active, uniquement si certaines tables pour le compte et le point de terminaison n'ont pas été renvoyées. Cette valeur n'existe pas dans une réponse si tous les noms de table sont déjà renvoyés. Utilisez cette valeur en tant que `ExclusiveStartTableName` dans une nouvelle demande pour continuer la liste jusqu'à ce que tous les noms de table soient renvoyés. Type : String

Erreurs spéciales

Il n'existe pas d'erreur spécifique de cette opération.

Exemples

Les exemples suivants illustrent une requête HTTP POST et une réponse utilisant l'opération ListTables.

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

Actions connexes

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Crée un élément ou remplace un ancien élément par un nouveau (en incluant tous les attributs). S'il existe déjà un élément dans la table spécifiée avec la même clé primaire, le nouvel élément remplace complètement l'élément existant. Vous pouvez effectuer une insertion conditionnelle (insérer un nouvel élément s'il n'existe pas de clé primaire spécifiée), ou remplacer un élément existant s'il possède certaines valeurs d'attribut.

Les valeurs d'attribut ne peuvent pas être nulles, les attributs de type chaîne et binaire doivent avoir une longueur supérieure à zéro, et les attributs de type ensemble ne peuvent pas être vides. Les demandes comprenant des valeurs vides sont rejetées avec une erreur `ValidationException`.

> **Note**
>
> Pour vous assurer qu'un nouvel élément ne remplace pas un élément existant, utilisez une opération d'insertion conditionnelle avec une valeur `Exists` définie sur `false` pour l'attribut de clé primaire ou les attributs.

Pour plus d'informations sur l'utilisation de `PutItem`, consultez Utilisation d'éléments et d'attributs dans DynamoDB.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table devant contenir l'élément. Type : String	Oui
`Item`	Mappage des attributs pour l'élément, qui doit inclure les valeurs de clé primaire qui définissent l'élément. D'autres paires nom-valeur d'attribu t peuvent être fournies pour l'élément. Pour plus d'informa tions sur les clés primaires, consultez Clé primaire. Type : mappage de noms d'attribut à des valeurs d'attribut.	Oui
`Expected`	Désigne un attribut pour une insertion conditionnelle. Le paramètre `Expected` vous permet de fournir un nom d'attribut et de spécifier si DynamoDB doit ou non vérifier si la valeur d'attribut existe, ou existe et a une valeur particuli ère, avant de modifier l'attribut. Type : mappage d'un nom d'attribut à une valeur d'attribut et vérification de l'existence de l'attribut.	Non
`Expected:AttributeName`	Nom de l'attribut pour l'inserti on conditionnelle.	Non

Name (Nom)	Description	Obligatoire
	Type : String	

Name (Nom)	Description	Obligatoire

`Expected:AttributeName: ExpectedAttributeValue`	Utilisez ce paramètre pour spécifier si une valeur existe déjà pour la paire nom-valeur de l'attribut.
	
	La notation JSON suivante remplace l'élément si l'attribut « Color » (Couleur) n'existe pas pour cet élément :
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La notation JSON suivante vérifie si l'attribut « Color » (Couleur) a une valeur existante de « Yellow » (Jaune) avant de remplacer l'élément :
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	
	Par défaut, si vous utilisez le paramètre `Expected` et fournissez une `Value`, DynamoDB suppose que l'attribut existe et a une valeur actuelle à remplacer. Vous n'avez donc pas à spécifier `{"Exists":true}` , car c'est implicite. Vous pouvez raccourcir la demande pour :
		Non

Name (Nom)	Description	Obligatoire
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Si vous spécifiez `{"Exists":true}` sans valeur d'attribut à vérifier, DynamoDB renvoie une erreur.	
`ReturnValues`	Utilisez ce paramètre si vous souhaitez obtenir les paires nom-valeur d'attribut avant leur mise à jour avec la demande `PutItem`. Les valeurs de paramètre possibles sont `NONE` (par défaut) ou `ALL_OLD`. Si `ALL_OLD` est spécifié et si l'opération `PutItem` a remplacé une paire nom-valeur d'attribut, le contenu de l'ancien élément est renvoyé. Si ce paramètre n'est pas fourni ou est `NONE`, rien n'est retourné. Type : String	Non

Réponses

Syntaxe

L'exemple de syntaxe suivant part du principe que la demande a spécifié un paramètre `ReturnValues` dont la valeur est `ALL_OLD`. Sinon, la réponse ne contient que l'élément `ConsumedCapacityUnits`.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

Nom	Description
`Attributes`	Valeurs d'attribut avant l'opération d'insertion, mais uniquement si le paramètre `ReturnValues` a la valeur `ALL_OLD` dans la demande. Type : mappage de paires nom-valeur d'attribut.
`ConsumedCapacityUnits`	Nombre d'unités de capacité d'écriture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Erreur	Description
`ConditionalCheckFailedException`	Le contrôle conditionnel a échoué. La valeur d'attribut attendue n'a pas été trouvée.
`ResourceNotFoundException`	L'élément ou l'attribut spécifiés n'ont pas été trouvés.

Exemples

Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation d'éléments et d'attributs dans DynamoDB.

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

Exemple de réponse

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84
```

```
{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

Actions connexes

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Requête

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Une opération `Query` obtient les valeurs d'un ou de plusieurs éléments et de leurs attributs par clé primaire (`Query` n'est disponible que pour des tables de clés primaires de hachage et de plage). Vous devez indiquer une `HashKeyValue` spécifique, et pouvez restreindre la portée de la requête à l'aide d'opérateurs de comparaison sur la `RangeKeyValue` de la clé primaire. Utilisez le paramètre `ScanIndexForward` pour obtenir les résultats dans l'ordre croissant ou inverse par clé de plage.

Les requêtes qui ne renvoient pas de résultat consomment les unités de capacité de lecture minimum en fonction du type de lecture.

> **Note**
>
> Si le nombre total d'éléments correspondant aux paramètres de requête dépasse la limite de 1 Mo, la requête s'arrête et les résultats sont renvoyés à l'utilisateur avec une `LastEvaluatedKey` pour continuer la requête dans une opération subséquente. Contrairement à une opération d'analyse, une opération de requête ne renvoie jamais un ensemble de résultats vide et une valeur `LastEvaluatedKey`. La valeur `LastEvaluatedKey` n'est fournie que si les résultats dépassent 1 Mo ou si vous avez utilisé le paramètre `Limit`.
> Le résultat peut être défini pour une lecture cohérente à l'aide du paramètre `ConsistentRead`.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table contenant les éléments demandés. Type : String	Oui
`AttributesToGet`	Tableau de noms d'attribut. Si des noms d'attribut ne sont pas spécifiés, tous les attributs sont renvoyés. Si certains attributs ne sont pas trouvés, ils n'apparaissent pas dans le résultat. Type : Array	Non
`Limit`	Nombre maximum d'éléments à renvoyer (pas nécessairement le nombre d'éléments correspondants). Si DynamoDB traite le nombre d'éléments jusqu'à la limite lors de l'interrogation de la table, il arrête la requête et renvoie les valeurs correspondantes jusqu'à ce point, ainsi qu'une valeur `LastEvaluatedKey` à appliquer dans une opération subséquente pour continuer la requête. Par ailleurs, si la taille de l'ensemble de résultats dépasse 1 Mo avant que DynamoDB atteigne cette limite, DynamoDB arrête la requête et renvoie les valeurs correspondantes, ainsi	Non

Name (Nom)	Description	Obligatoire
	qu'une valeur `LastEvaluatedKey` à appliquer dans une opération subséquente pour continuer la requête. Type : nombre	
`ConsistentRead`	Si la valeur est définie sur `true`, une lecture cohérente est effectuée. Sinon une cohérence éventuelle est utilisée. Type : booléen	Non

Name (Nom)	Description	Obligatoire
`Count`	Si la valeur est définie sur `true`, DynamoDB renvoie un nombre total d'éléments qui correspondent aux paramètres de requête, au lieu d'une liste des éléments correspondants et de leurs attributs. Vous pouvez appliquer le paramètre `Limit` aux requêtes de nombre uniquement. Ne pas définissez pas la valeur `Count` sur `true` tout en fournissant une liste `AttributesToGet` . Sinon, DynamoDB renvoie une erreur de validation. Pour de plus amples informati ons, consultez Comptabil isation des éléments dans les résultats. Type : booléen	Non
`HashKeyValue`	Valeur d'attribut du composant de hachage de la clé primaire composite. Type : chaîne, nombre ou binaire	Oui

Name (Nom)	Description	Obligatoire
`RangeKeyCondition`	Conteneur pour les valeurs d'attribut et les opérateurs de comparaison à utiliser pour la requête. Une demande de requête ne nécessite pas de `RangeKeyCondition` . Si vous fournissez uniquement la valeur `HashKeyValue` , DynamoDB renvoie tous les éléments avec la valeur d'élément de clé de hachage spécifiée. Type : carte	Non
`RangeKeyCondition` : `AttributeValueList`	Valeurs d'attribut à évaluer pour les paramètres de requête. La `AttributeValueList` contient une valeur d'attribut, sauf si une comparaison `BETWEEN` est spécifiée. Pour la comparaison `BETWEEN`, la `AttributeValueList` contient deux valeurs d'attribut. Type : mappage de `AttributeValue` à un `ComparisonOperator` .	Non

Name (Nom)	Description	Obligatoire
`RangeKeyCondition : ComparisonOperator`	Critères d'évaluation des attributs fournis, tels que Egal à, Supérieur à, etc. Les opérateurs de comparaison valides pour une opération de requête sont les suivants. **Note** Les comparaisons de valeurs de chaîne pour les critères Supérieur à, Egal à ou Inférieur à sont basées sur des valeurs de code de caractère ASCII. Par exemple, `a` est supérieur à `A`, et `aa` est supérieur à `B`. Pour obtenir la liste des valeurs des codes, consultez http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange. Pour une valeur binaire, DynamoDB traite chaque octet des données binaires comme non signé quand il compare des valeurs binaires, par exemple, lors de	Non

Name (Nom)	Description	Obligatoire
	l'évaluation d'expressions de requête. Type : chaîne ou binaire	
	`EQ` : égal. Pour `EQ`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`LE` : inférieur ou égal. Pour `LE`, `Attribute ValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`LT` : inférieur. Pour `LT`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`GE` : supérieur ou égal. Pour `GE`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`GT` : supérieur. Pour `GT`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.	
	`BEGINS_WITH` : recherche un préfixe. Pour `BEGINS_WITH` , `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type chaîne ou binaire (pas nombre ou ensemble). L'attribut cible de la comparaison doit être une chaîne ou un binaire (pas un nombre ou un ensemble).	

Name (Nom)	Description	Obligatoire
	`BETWEEN` : supérieur ou égal à la première valeur, et inférieur ou égal à la deuxième valeur. Pour `BETWEEN`, `Attribute ValueList` doit inclure deux éléments `Attribute Value` du même type, Chaîne, Nombre ou Binaire (et non Ensemble). Un attribut cible correspond si la valeur cible est supérieure ou égale au premier élément, et inférieure ou égale au deuxième. Si un élément contient une `Attribute Value` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas comparable à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}` .	

Name (Nom)	Description	Obligatoire
`ScanIndexForward`	Spécifie le balayage ascendant ou descendant de l'index. DynamoDB renvoie des résultats reflétant l'ordre demandé déterminé par la clé de plage. Si le type de données est Nombre, les résultats sont renvoyés dans l'ordre numérique. Sinon, le balayage est basé sur des valeurs de code de caractère ASCII. Type : booléen Le paramétrage par défaut est `true` (ascendant).	Non

Name (Nom)	Description	Obligatoire
`ExclusiveStartKey`	Clé primaire de l'élément à partir duquel continuer une requête antérieure. Une requête antérieure peut fournir cette valeur en tant que `LastEvaluatedKey` si cette opération de requête a été interrompue avant la fin, soit en raison de la taille de l'ensemble de résultats, soit en raison du paramètre `Limit`. La `LastEvaluatedKey` peut être retransmise dans une nouvelle demande de requête pour continuer l'opération à partir de ce point. Type : `HashKeyElement` , ou `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.	Non

Réponses

Syntaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"N":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
```

```
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

Name (Nom)	Description
`Items`	Attributs d'élément répondant aux paramètres de requête. Type : mappage de noms d'attribut à leurs types de données et valeurs.
`Count`	Nombre d'éléments dans la réponse. Pour de plus amples informations, consultez Comptabilisation des éléments dans les résultats. Type : nombre
`LastEvaluatedKey`	Clé primaire de l'élément où l'opération de requête s'est arrêtée, incluant l'ensemble de résultats précédent. Utilisez cette valeur pour démarrer une nouvelle opération excluant cette valeur dans la nouvelle demande. La `LastEvaluatedKey` a une valeur `null` quand l'ensemble des résultats de la requête est complet (c'est-à-dire que l'opération a traité la « dernière page »). Type : `HashKeyElement`, ou `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.

Name (Nom)	Description
`ConsumedCapacityUnits`	Nombre d'unités de capacité de lecture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Erreur	Description
`ResourceNotFoundException`	La table spécifiée n'a pas été trouvée.

Exemples

Pour des exemples d'utilisation du kit SDK AWS, consultez Interrogation de tables dans DynamoDB.

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
```

```
}
```

Exemple de réponse

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

Exemple de réponse

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

Actions connexes

- Analyser

Analyser

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

L'opération `Scan` renvoie un ou plusieurs éléments et leurs attributs en effectuant une analyse complète d'une table. Fournissez un `ScanFilter` pour obtenir des résultats plus spécifiques.

> **Note**
>
> Si le nombre total d'éléments analysés dépasse la limite de 1 Mo, l'analyse s'arrête et les résultats sont renvoyés à l'utilisateur avec une `LastEvaluatedKey` pour continuer l'analyse dans une opération subséquente. Les résultats incluent également le nombre d'éléments

dépassant la limite. Il peut arriver qu'une analyse constate l'absence de données de table répondant aux critères de filtre.
L'ensemble de résultats est éventuellement cohérent.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table contenant les éléments demandés. Type : String	Oui
`AttributesToGet`	Tableau de noms d'attribut. Si des noms d'attribut ne sont pas spécifiés, tous les attributs sont renvoyés. Si certains attributs ne sont pas trouvés,	Non

Name (Nom)	Description	Obligatoire
	ils n'apparaissent pas dans le résultat. Type : Array	
`Limit`	Nombre maximum d'éléments à évaluer (pas nécessairement le nombre d'éléments correspondants). Si DynamoDB traite le nombre d'éléments jusqu'à la limite lors du traitement des résultats, il arrête et renvoie les valeurs correspondantes jusqu'à ce point, ainsi qu'une valeur `LastEvaluatedKey` à appliquer dans une opération subséquente pour continuer à récupérer des éléments. Par ailleurs, si la taille du jeu de données dépasse 1 Mo avant que DynamoDB atteigne cette limite, DynamoDB arrête l'analyse et renvoie les valeurs correspondantes jusqu'à la limite, ainsi qu'une valeur `LastEvaluatedKey` à appliquer dans une opération subséquente pour continuer l'analyse. Type : nombre	Non

Name (Nom)	Description	Obligatoire
`Count`	Si la valeur est définie sur `true`, DynamoDB renvoie un nombre total d'éléments pour l'opération d'analyse, même si celle-ci ne trouve pas d'élément pour le filtre affecté. Vous pouvez appliquer le paramètre Limit aux analyses de nombre uniquement. Ne définissez pas la valeur `Count` sur `true` tout en fournissant une liste `AttributesToGet` car, dans ce cas, DynamoDB renvoie une erreur de validation. Pour de plus amples informations, consultez Comptabilisation des éléments dans les résultats. Type : booléen	Non

Name (Nom)	Description	Obligatoire
`ScanFilter`	Evalue les résultats de l'analyse et renvoie uniquement les valeurs souhaitées. S'il y a plusieurs conditions, celles-ci sont traitées comme des opérations « AND (ET) ». Toutes les conditions doivent être remplies pour être incluses dans les résultats. Type : mappage des noms d'attribut à des valeurs avec des opérateurs de comparaison.	Non
`ScanFilter` :`AttributeValueList`	Valeurs et conditions d'évaluation des résultats d'analyse pour le filtre. Type : mappage de `AttributeValue` à un `Condition`.	Non

Name (Nom)	Description	Obligatoire
`ScanFilter : ComparisonOperator`	Critères d'évaluation des attributs fournis, tels que Egal à, Supérieur à, etc. Les opérateurs de comparaison valides pour une opération d'analyse sont les suivants. **Note** Les comparaisons de valeurs de chaîne pour les critères Supérieur à, Egal à ou Inférieur à sont basées sur des valeurs de code de caractère ASCII. Par exemple, `a` est supérieur à `A`, et `aa` est supérieur à `B`. Pour obtenir la liste des valeurs des codes, consultez http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange. Pour une valeur binaire, DynamoDB traite chaque octet des données binaires comme non signé quand il compare des valeurs binaires, par exemple, lors de	Non

Name (Nom)	Description	Obligatoire
	l'évaluation d'expressions de requête. Type : chaîne ou binaire	
	`EQ` : égal. Pour `EQ`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	NE : non égal. Pour NE, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`LE` : inférieur ou égal. Pour `LE`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`LT` : inférieur. Pour `LT`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`GE` : supérieur ou égal. Pour `GE`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.	

Name (Nom)	Description	Obligatoire
	`GT` : supérieur. Pour `GT`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.	
	`NOT_NULL` : l'attribut existe.	
	`NULL` : l'attribut n'existe pas.	

Name (Nom)	Description	Obligatoire
	`CONTAINS` : vérifie la présence d'une sous-séquence ou d'une valeur dans un ensemble. Pour `CONTAINS`, `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si l'attribut cible de la comparaison est de type chaîne, l'opération vérifie la présence d'une sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type binaire, l'opération vérifie la présence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est de type ensemble (« SS », « NS » ou « BS »), l'opération vérifie la présence d'un membre de l'ensemble (pas comme une sous-chaîne).	

Name (Nom)	Description	Obligatoire
	`NOT_CONTAINS` : vérifie l'absence d'une sous-séquence ou d'une valeur dans un ensemble. Pour `NOT_CONTAINS` , `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Si l'attribut cible de la comparaison est de type chaîne, l'opération vérifie l'absence de sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type binaire, l'opération vérifie l'absence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est de type ensemble (« SS », « NS » ou « BS »), l'opération vérifie l'absence d'un membre de l'ensemble (pas comme une sous-chaîne).	

Name (Nom)	Description	Obligatoire
	`BEGINS_WITH` : vérifie la présence d'un préfixe. Pour `BEGINS_WITH` , `AttributeValueList` ne peut contenir qu'une seule `AttributeValue` de type chaîne ou binaire (pas nombre ou ensemble). L'attribut cible de la comparaison doit être une chaîne ou un binaire (pas un nombre ou un ensemble).	
	`IN` : vérifie la présence de correspondances exactes. Pour `IN`, `AttributeValueList` peut contenir qu'une seule `AttributeValue` de type Chaîne, Nombre ou Binaire (et non Ensemble). Pour correspondre, l'attribut cible de la comparaison doit être du même type et avoir exactement la même valeur. Une chaîne ne correspond jamais à un ensemble de chaîne.	

Name (Nom)	Description	Obligatoire
	`BETWEEN` : supérieur ou égal à la première valeur, et inférieur ou égal à la deuxième valeur. Pour `BETWEEN`, `Attribute ValueList` doit inclure deux éléments `Attribute Value` du même type, Chaîne, Nombre ou Binaire (et non Ensemble). Un attribut cible correspond si la valeur cible est supérieure ou égale au premier élément, et inférieure ou égale au deuxième. Si un élément contient une `Attribute Value` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas comparable à `{"N":"6"}` . De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}` .	

Name (Nom)	Description	Obligatoire
`ExclusiveStartKey`	Clé primaire de l'élément à partir duquel continuer une analyse antérieure. Une analyse antérieure peut fournir cette valeur si cette opération d'analyse a été interrompue avant la fin, soit en raison de la taille de l'ensemble de résultats, soit en raison du paramètre `Limit`. La `LastEvaluatedKey` peut être retransmise dans une nouvelle demande d'analyse pour continuer l'opération à partir de ce point. Type : `HashKeyElement` , ou `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.	Non

Réponses

Syntaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue4"},
```

```
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

Name (Nom)	Description
`Items`	Conteneur pour les attributs répondant aux paramètres de l'opération. Type : mappage de noms d'attribut à leurs types de données et valeurs.
`Count`	Nombre d'éléments dans la réponse. Pour de plus amples informations, consultez Comptabilisation des éléments dans les résultats. Type : nombre
`ScannedCount`	Nombre d'éléments dans l'analyse complète avant l'application des filtres. Une valeur `ScannedCount` élevée avec peu ou pas de résultats `Count` indique une opération d'analyse inefficace. Pour de plus amples informations, consultez Comptabilisation des éléments dans les résultats. Type : nombre
`LastEvaluatedKey`	Clé primaire de l'élément où l'opération d'analyse s'est arrêtée. Fournissez cette valeur dans une opération d'analyse subséquente pour poursuivre l'opération à partir de ce point.

Name (Nom)	Description
	La `LastEvaluatedKey` a une valeur `null` quand l'ensemble des résultats de l'analyse est complet (c'est-à-dire que l'opération a traité la « dernière page »).
`ConsumedCapacityUnits`	Nombre d'unités de capacité de lecture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Erreur	Description
`ResourceNotFoundException`	La table spécifiée n'a pas été trouvée.

Exemples

Pour des exemples d'utilisation du kit SDK AWS, consultez Analyse de tables dans DynamoDB.

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Exemple de réponse

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"}
 },{
 "date":{"S":"1999"},
 "fans":{"SS":["Ziggy","Laura","Dean"]},
 "name":{"S":"Matrix"},
 "rating":{"S":"*****"}
 },{
 "date":{"S":"1976"},
 "fans":{"SS":["Riley"]},"
 name":{"S":"The Shaggy D.A."},
 "rating":{"S":"**"}
 },{
 "date":{"S":"1985"},
 "fans":{"SS":["Fox","Lloyd"]},
 "name":{"S":"Back To The Future"},
 "rating":{"S":"****"}
 }],
    "ConsumedCapacityUnits":0.5
 "ScannedCount":4}
```

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
```

```
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
```

```
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
"ScannedCount":2
}
```

Actions connexes

- Requête
- BatchGetItem

UpdateItem

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Modifie les attributs d'un élément existant. Vous pouvez effectuer une mise à jour conditionnelle (insérer une nouvelle paire nom-valeur d'attribut si elle n'existe pas, ou remplacer une paire nom-valeur existante si elle a certaines valeurs d'attribut attendues).

> **Note**
>
> Vous ne pouvez pas mettre à jour les attributs de clé primaire en utilisant l'opération UpdateItem. Au lieu de cela, supprimez l'élément et utilisez l'opération PutItem pour créer un nouvel élément avec de nouveaux attributs.

L'opération UpdateItem inclut un paramètre `Action` qui définit comment effectuer la mise à jour. Vous pouvez insérer, supprimer ou ajouter des valeurs d'attribut.

Les valeurs d'attribut ne peuvent pas être nulles, les attributs de type chaîne et binaire doivent avoir une longueur supérieure à zéro, et les attributs de type ensemble ne peuvent pas être vides. Les demandes comprenant des valeurs vides sont rejetées avec une erreur `ValidationException`.

Si un élément existant possède la clé primaire spécifiée :

- PUT – Ajoute l'attribut spécifié. Si l'attribut existe, il est remplacé par la nouvelle valeur.
- DELETE – Si aucune valeur n'est spécifiée, l'attribut et sa valeur sont supprimés. Si un ensemble de valeurs est spécifié, les valeurs dans l'ensemble spécifié sont supprimées de l'ancien ensemble. Ainsi, si la valeur d'attribut contient [a,b,c] et si l'action de suppression contient [a,c], la valeur d'attribut finale est [b]. Le type de la valeur spécifiée doit correspondre au type de valeur existant. La spécification d'un ensemble vide n'est pas valide.
- ADD – N'utilisez l'action d'ajout que pour des nombres ou si l'attribut cible est un ensemble (y compris un ensemble de chaîne). L'action ADD ne fonctionne pas si l'attribut cible est une valeur de chaîne unique ou une valeur binaire scalaire. La valeur spécifiée est ajoutée à une valeur numérique (incrémentant ou décrémentant la valeur numérique existante), ou ajoutée en tant que valeur supplémentaire dans un ensemble de chaîne. Si un ensemble de valeurs est spécifié, les valeurs sont ajoutées à l'ensemble existant. Par exemple, si l'ensemble d'origine est [1,2] et si la valeur fournie est [3], après l'opération d'ajout, l'ensemble est [1,2,3], non [4,5]. Une erreur se produit si une action d'ajout est spécifiée pour un attribut d'ensemble et si le type d'attribut spécifié ne correspond pas au type d'ensemble existant.

Si vous utilisez l'action ADD pour un attribut qui n'existe pas, l'attribut et ses valeurs sont ajoutés à l'élément.

Si aucun élément ne correspond à la clé primaire spécifiée :

- PUT – Crée un élément avec la clé primaire spécifiée. Ajoute ensuite l'attribut spécifié.
- DELETE – Il ne se passe rien.
- ADD – Crée un élément avec la clé primaire et le numéro (ou un ensemble de numéros) fournis pour la valeur d'attribut. Non valide pour un type chaîne ou binaire.

> **Note**
>
> Si vous utilisez l'action ADD pour incrémenter ou décrémenter une valeur numérique pour un élément qui n'existe pas avant la mise à jour, DynamoDB utilise `0` comme valeur initiale. Par ailleurs, si vous mettez à jour un élément en utilisant l'action ADD pour incrémenter ou décrémenter une valeur numérique pour un attribut qui n'existe pas avant la mise à jour (tandis que l'élément existe), DynamoDB utilise `0` comme valeur initiale. Par exemple, vous utilisez l'action ADD pour ajouter `+3` à un attribut qui n'existait pas avant la mise à jour. DynamoDB utilise `0` comme valeur initiale, et la valeur après la mise à jour est `3`.

Pour plus d'informations sur l'utilisation de cette opération, consultez Utilisation d'éléments et d'attributs dans DynamoDB.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
```

```
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table contenant l'élément à mettre à jour. Type : String	Oui
`Key`	Clé primaire définissant l'élément. Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : mappage de `HashKeyElement` à sa valeur, et de `RangeKeyElement` à sa valeur.	Oui
`AttributeUpdates`	Mappage de nom d'attribut à la nouvelle valeur, et d'une action pour la mise à jour. Les noms d'attribut spécifient les attributs à modifier, et ne peuvent pas contenir d'attributs de clé primaire. Type : mappage de nom d'attribut, de valeur et d'une action pour la mise à jour.	
`AttributeUpdates :Action`	Spécifie comment effectuer la mise à jour. Valeurs possibles : `PUT` (par défaut), `ADD` ou `DELETE`. La sémantiqu	Non

Name (Nom)	Description	Obligatoire
	e est expliquée dans la description de l'opération UpdateItem. Type : String Par défaut : PUT	
`Expected`	Désigne un attribut pour une mise à jour conditionnelle. Le paramètre `Expected` vous permet de fournir un nom d'attribut et de spécifier si DynamoDB doit ou non vérifier si la valeur d'attribut existe, ou existe et a une valeur particuli ère, avant de modifier l'attribut. Type : mappage de noms d'attribut.	Non
`Expected:AttributeName`	Nom de l'attribut pour l'inserti on conditionnelle. Type : String	Non

Name (Nom)	Description	Obligatoire

`Expected:AttributeName: ExpectedAttributeValue`	Utilisez ce paramètre pour spécifier si une valeur existe déjà pour la paire nom-valeur de l'attribut.
	
	La notation JSON suivante met à jour l'élément si l'attribut « Color » (Couleur) n'existe pas pour cet élément :
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La notation JSON suivante vérifie si l'attribut « Color » (Couleur) a une valeur existante de « Yellow » (Jaune) avant de mettre à jour l'élément :
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	Par défaut, si vous utilisez le paramètre `Expected` et fournissez une `Value`, DynamoDB suppose que l'attribut existe et a une valeur actuelle à remplacer. Vous n'avez donc pas à spécifier `{"Exists":true}` , car c'est implicite. Vous pouvez raccourcir la demande pour :
		Non

Name (Nom)	Description	Obligatoire
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Si vous spécifiez `{"Exists":true}` sans valeur d'attribut à vérifier, DynamoDB renvoie une erreur.	

Name (Nom)	Description	Obligatoire
`ReturnValues`	Utilisez ce paramètre si vous souhaitez obtenir les paires nom-valeur d'attribut avant leur mise à jour avec la demande `UpdateItem` . Les valeurs de paramètre possibles sont `NONE` (par défaut) ou `ALL_OLD`, `UPDATED_OLD` , `ALL_NEW` ou `UPDATED_NEW` . Si `ALL_OLD` est spécifié et si l'opération `UpdateItem` a remplacé une paire nom-valeur d'attribut, le contenu de l'ancien élément est renvoyé. Si ce paramètre n'est pas fourni ou est `NONE`, rien n'est retourné. Si `ALL_NEW` est spécifié, tous les attributs de la nouvelle version de l'élément sont renvoyés. Si `UPDATED_NEW` est spécifié, les nouvelles versions uniquement des attributs mis à jour sont renvoyées. Type : String	Non

Réponses

Syntaxe

L'exemple de syntaxe suivant part du principe que la demande a spécifié un paramètre `ReturnValues` dont la valeur est `ALL_OLD`. Sinon, la réponse ne contient que l'élément `ConsumedCapacityUnits`.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

Nom	Description
`Attributes`	Mappage de paires nom-valeur d'attribut, mais uniquement si le paramètre `ReturnValues` est spécifié comme autre chose que `NONE` dans la demande. Type : mappage de paires nom-valeur d'attribut.
`ConsumedCapacityUnits`	Nombre d'unités de capacité d'écriture consommées par l'opération. Cette valeur indique le nombre appliqué à votre débit approvisionné. Pour plus d'informations, consultez Mode de capacité provisionnée DynamoDB. Type : nombre

Erreurs spéciales

Erreur	Description
`ConditionalCheckFailedException`	Le contrôle conditionnel a échoué. La valeur (« + name + ») de l'attribut est (« + value + ») mais la valeur attendue était (« + ExpValue + »)
`ResourceNotFoundExceptions`	L'élément ou l'attribut spécifiés n'ont pas été trouvés.

Exemples

Pour des exemples d'utilisation du kit SDK AWS, consultez Utilisation d’éléments et d’attributs dans DynamoDB.

Exemple de demande

```
// This header is abbreviated. For a sample of a complete header, see API de bas niveau
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
```

```
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
"ConsumedCapacityUnits":1
}
```

Actions connexes

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***Cette section fait référence à l'API version 2011-12-05 qui est obsolète et ne doit pas être utilisée pour de nouvelles applications.***
> Pour une documentation sur l'API de bas niveau actuelle, consultez la Référence d'API Amazon DynamoDB.

Description

Met à jour le débit approvisionné pour la table donnée. La définition du débit pour une table vous aide à gérer les performances et fait partie de la fonction de débit approvisionné de DynamoDB. Pour de plus amples informations, consultez Mode de capacité provisionnée DynamoDB.

Les valeurs de débit approvisionné peuvent faire l'objet d'une mise à niveau en fonction des maxima et minima répertoriés dans Quotas dans Amazon DynamoDB.

Pour que cette opération réussisse, la table doit être dans l'état `ACTIVE`. L'opération UpdateTable est asynchrone. Lors de son exécution, la table est dans l'état `UPDATING`. Quand la table est dans l'état `UPDATING`, elle a encore le débit approvisionné d'avant l'appel. Le nouveau paramètre de débit approvisionné ne prend effet que quand la table revient à l'état `ACTIVE` après l'opération UpdateTable.

Requêtes

Syntaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Name (Nom)	Description	Obligatoire
`TableName`	Nom de la table à mettre à jour. Type : String	Oui
`ProvisionedThroughput`	Nouveau débit pour la table spécifiée, composé de valeurs pour `ReadCapacityUnits` et `WriteCapacityUnits` . Consultez Mode de capacité provisionnée DynamoDB. Type : Array	Oui
`ProvisionedThroughput :ReadCapacityUnits`	Définit le nombre minimum de `ReadCapacityUnits` cohérentes consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations.	Oui

Name (Nom)	Description	Obligatoire
	Des opérations de lecture éventuellement cohérente nécessitant moins d'effort qu'une opération de lecture cohérente, un paramètre de 50 `ReadCapacityUnits` par seconde fournit 100 `ReadCapacityUnits` éventuellement cohérentes par seconde. Type : nombre	
`ProvisionedThroughput :WriteCapacityUnits`	Définit le nombre minimum de `WriteCapacityUnits` consommées par seconde pour la table spécifiée avant que DynamoDB équilibre la charge avec d'autres opérations. Type : nombre	Oui

Réponses

Syntaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
```

```
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

Name (Nom)	Description
`CreationDateTime`	Date de création de la table. Type : nombre
`KeySchema`	Structure (simple ou composite) de la clé primaire pour la table. Une paire nom-valeur pour l'élément `HashKeyElement` est obligatoire, et une paire nom-valeur pour l'élément `RangeKeyElement` est facultatif (obligatoire uniquement pour les clés primaires composites). La taille maximum de clé de hachage est de 2 048 octets. La taille maximum de clé de plage est de 1 024 octets. Les deux limites sont appliquées séparément (c'est-à-dire que vous pouvez avoir une taille de clé combinée de hachage et de plage égale à 2 048 + 1 024 octets). Pour plus d'informations sur les clés primaires, consultez Clé primaire. Type : mappage de `HashKeyElement`, ou de `HashKeyElement` et `RangeKeyElement` pour une clé primaire composite.
`ProvisionedThroughput`	Paramètres de débit actuels pour la table spécifiée, incluant des valeurs pour `LastIncreaseDateTime` (le cas échéant), `LastDecreaseDateTime` (le cas échéant),

Name (Nom)	Description
	Type : Array
`TableName`	Nom de la table mise à jour. Type : String
`TableStatus`	État actuel de la table (`CREATING`, `ACTIVE`, `DELETING` ou `UPDATING`), qui devrait être `UPDATING`. Utilisez l'opération DescribeTables pour vérifier l'état de la table. Type : String

Erreurs spéciales

Erreur	Description
`ResourceNotFoundException`	La table spécifiée n'a pas été trouvée.
`ResourceInUseException`	La table n'est pas dans l'état `ACTIVE`.

Exemples

Exemple de demande

```
// This header is abbreviated.
// For a sample of a complete header, see API de bas niveau de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Exemple de réponse

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
    "TableStatus":"UPDATING"}
}
```

Actions connexes

- CreateTable
- DescribeTables
- DeleteTable

Paramètres conditionnels hérités de DynamoDB

Ce document fournit une vue d'ensemble des paramètres conditionnels hérités de DynamoDB et recommande d'utiliser les nouveaux paramètres d'expression à la place. Il fournit des informations détaillées sur des paramètres tels que AttributesToGet, AttributeUpdates, ConditionalOperator, Expected, KeyConditions, QueryFilter et ScanFilter, ainsi que des exemples d'utilisation des nouveaux paramètres d'expression en remplacement.

> **Important**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB.
>
> De plus, DynamoDB ne permet pas de mélanger des paramètres conditionnels hérités et des paramètres d'expression dans un même appel. Par exemple, l'appel de l'opération `Query` avec `AttributesToGet` et `ConditionExpression` entraîne une erreur.

Le tableau suivant présente les opérations d'API DynamoDB qui prennent encore en charge ces paramètres hérités ainsi que les paramètres d'expression à utiliser à leur place. Ce tableau peut être utile si vous envisagez de mettre à jour vos applications afin qu'elles utilisent plutôt des paramètres d'expression.

Si vous utilisez cette opération d'API...	Avec ces paramètres hérités…	Utilisez plutôt ce paramètre d'expression
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`

Si vous utilisez cette opération d'API...	Avec ces paramètres hérités…	Utilisez plutôt ce paramètre d'expression
	`Expected`	`ConditionExpression`

Les sections suivantes fournissent des informations supplémentaires sur les paramètres conditionnels hérités.

Rubriques

- AttributesToGet (hérité)
- AttributeUpdates (hérité)
- ConditionalOperator (hérité)
- Expected (hérité)
- KeyConditions (hérité)
- QueryFilter (hérité)
- ScanFilter (hérité)
- Conditions d'écriture avec des paramètres hérités

AttributesToGet (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, Utiliser ProjectionExpression à la place.

Le paramètre conditionnel hérité `AttributesToGet` est un tableau d'un ou plusieurs attributs à extraire de DynamoDB. Si aucun nom d'attribut n'est fourni, tous les attributs sont renvoyés. Les attributs demandés non trouvés n'apparaissent pas dans le résultat.

`AttributesToGet` vous permet d'extraire des attributs de type liste ou mappage, mais pas des éléments individuels d'une liste ou d'un mappage.

Notez qu'`AttributesToGet` n'a aucun effet sur la consommation du débit approvisionné. DynamoDB détermine les unités de capacité consommées sur la base de la taille d'un élément, pas de la quantité de données qu'il renvoie à une application.

Utiliser ProjectionExpression à la place – Exemple

Supposons que vous souhaitez extraire un élément de la table Music, mais ne voulez renvoyer que certains des attributs. Vous pourriez utiliser une requête `GetItem` avec un paramètre `AttributesToGet` comme dans cet exemple d'AWS CLI :

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

Vous pouvez utiliser `ProjectionExpression` à la place :

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, utilisez UpdateExpression à la place..

Dans une opération `UpdateItem`, le paramètre conditionnel hérité `AttributeUpdates` contient les noms d'attributs à modifier, l'action à effectuer sur chacun d'eux et leur nouvelle valeur. Si vous

mettez à jour un attribut de clé d'index pour tous les index figurant dans cette table, le type d'attribut doit correspondre au type de clé d'index défini dans la section `AttributesDefinition` de la description de la table. Vous pouvez utiliser une opération `UpdateItem` pour mettre à jour tous les attributs autres que de clé.

Les valeurs d'attribut ne peuvent pas être nulles. Les attributs de type chaîne et binaire doivent avoir une longueur supérieure à zéro. Les attributs de type ensemble ne peuvent pas être vides. Les demandes comprenant des valeurs vides sont rejetées avec une exception `ValidationException`.

Chaque élément `AttributeUpdates` se compose d'un nom d'attribut à modifier, ainsi que de ce qui suit :

- `Value` : nouvelle valeur, le cas échéant, pour cet attribut.
- `Action` : valeur spécifiant comment effectuer la mise à jour. Cette action n'est valide que pour un attribut existant dont le type de données est nombre ou ensemble. N'utilisez pas l'action `ADD` pour d'autres types de données.

  Si un élément avec la clé primaire spécifiée est trouvé dans la table, les valeurs suivantes exécutent les actions suivantes :

  - `PUT` - Ajoute l'attribut spécifié à l'élément. Si l'attribut existe déjà, il est remplacé par la nouvelle valeur.
  - `DELETE` - Si aucune valeur n'est spécifiée pour `DELETE`, supprime l'attribut et sa valeur. Le type de données de la valeur spécifiée doit correspondre au type de données de la valeur existante.

    Si un ensemble de valeurs est spécifié, ces valeurs sont soustraites de l'ancien ensemble. Par exemple, si la valeur d'attribut était l'ensemble `[a,b,c]` et si l'action `DELETE` spécifie `[a,c]`, la valeur d'attribut finale est `[b]`. La spécification d'un ensemble vide est une erreur.
  - `ADD` - Ajoute la valeur spécifiée à l'élément si l'attribut n'existe pas encore. Si l'attribut existe, le comportement de l'action `ADD` dépend du type de données de l'attribut :
    - Si l'attribut existant est un nombre et si `Value` est également un nombre, `Value` est ajouté mathématiquement à l'attribut existant. Si `Value` est un nombre négatif, celui-ci est soustrait de l'attribut existant.

> **Note**
>
> Si vous utilisez l'action `ADD` pour incrémenter ou décrémenter une valeur numérique pour un élément qui n'existe pas avant la mise à jour, DynamoDB utilise 0 comme valeur initiale.
>
> De même, si vous utilisez l'action `ADD` pour un élément existant afin d'incrémenter ou de décrémenter une valeur d'attribut qui n'existe pas avant la mise à jour, DynamoDB utilise `0` comme valeur initiale. Par exemple, supposons que l'élément que vous souhaitez mettre à jour ne possède pas d'attribut nommé itemcount, mais que vous décidez malgré tout d'ajouter (`ADD`) le nombre `3` à cet attribut. DynamoDB crée l'attribut itemcount, définit sa valeur initiale sur `0`, puis y ajoute `3`. Le résultat sera un nouvel attribut itemcount, avec une valeur de `3`.

- Si le type de données existant est un ensemble, et si `Value` est également un ensemble, `Value` est ajouté à l'ensemble existant. Par exemple, si la valeur d'attribut est l'ensemble `[1,2]` et si l'action `ADD` spécifie `[3]`, la valeur d'attribut finale est `[1,2,3]`. Une erreur se produit si une action `ADD` est spécifiée pour un attribut ensemble et si le type d'attribut spécifié ne correspond pas au type d'ensemble existant.

  Les deux ensembles doivent avoir le même type de données primitif. Par exemple, si le type de données existant est un ensemble de chaînes, `Value` doit également être un ensemble de chaînes.

Si aucun élément avec la clé primaire spécifiée n'est trouvé dans la table, les valeurs suivantes exécutent les actions suivantes :

- `PUT` - Amène DynamoDB à créer un élément avec la clé primaire spécifiée, puis ajoute l'attribut.
- `DELETE` - Rien ne se passe, car des attributs ne peuvent pas être supprimés d'un élément inexistant. L'opération réussit, mais DynamoDB ne crée pas d'élément.
- `ADD` – Amène DynamoDB à créer un élément avec la clé primaire et le nombre (ou ensemble de nombres) fournis pour la valeur d'attribut. Les seuls types de données autorisés sont Nombre et Ensemble de nombres.

Si vous fournissez des attributs faisant partie d'une clé d'index, les types de données pour ces attributs doivent correspondre à ceux du schéma dans la définition d'attribut de la table.

Utiliser UpdateExpression à la place – Exemple

Supposons que vous souhaitez modifier un élément dans la table Music. Vous pourriez utiliser une requête `UpdateItem` avec un paramètre `AttributeUpdates` comme dans cet exemple d'AWS CLI :

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

Vous pouvez utiliser `UpdateExpression` à la place :

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

ConditionalOperator (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB.

Le paramètre conditionnel hérité `ConditionalOperator` est un opérateur logique destiné s'appliquer aux conditions d'un mappage `Expected`, `QueryFilter` ou `ScanFilter` :

- AND - Si toutes les conditions ont la valeur true, le mappage entier a la valeur true.
- OR – Si au moins une des conditions a la valeur true, le mappage entier a la valeur true.

Si vous omettez `ConditionalOperator`, `AND` est l'opérateur par défaut.

L'opération ne réussit que si le mappage entier a la valeur true.

> **Note**
>
> Ce paramètre ne prend pas en charge les attributs de type Liste ou Mappage.

Expected (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, utilisez ConditionExpression à la place..

Le paramètre conditionnel hérité `Expected` est un bloc conditionnel pour une opération `UpdateItem`. `Expected` est un mappage de paires attribut/condition. Chaque élément du mappage se compose d'un nom d'attribut, d'un opérateur de comparaison et d'une ou plusieurs valeurs. DynamoDB compare l'attribut avec la ou les valeurs que vous avez fournies, en utilisant l'opérateur de comparaison. Pour chaque élément `Expected`, le résultat de l'évaluation est vrai (true) ou faux (false).

Si vous spécifiez plusieurs éléments dans le mappage `Expected`, par défaut, le résultat de toutes les conditions doit être true. En d'autres termes, les conditions sont combinées à l'aide de l'opérateur `AND`. Vous pouvez également utiliser le paramètre `ConditionalOperator` pour dissocier les conditions à l'aide de l'opérateur logique OR (OU). Dans ce cas, au moins une des conditions doit avoir la valeur vrai, plutôt que toutes.

Si le mappage `Expected` a la valeur true, l'opération conditionnelle réussit. Autrement, elle échoue.

Le mappage `Expected` contient les éléments suivants :

- `AttributeValueList` - Une ou plusieurs valeurs à évaluer par rapport à l'attribut fourni. Le nombre de valeurs dans la liste dépend de l'opérateur `ComparisonOperator` utilisé.

  Pour le type Nombre, les comparaisons de valeurs sont numériques.

  Les comparaisons de valeurs de chaîne pour supérieur, égal ou inférieur sont basées sur le codage Unicode binaire UTF-8. Par exemple, `a` est supérieur à `A`, et `a` est supérieur à `B`.

  Pour le type Binaire, DynamoDB traite chaque octet des données binaires comme non signé lors de la comparaison de valeurs binaires.
- `ComparisonOperator` - Comparateur pour l'évaluation d'attributs dans la `AttributeValueList`. Lors de la comparaison, DynamoDB utilise des lectures fortement cohérentes.

  Les opérateurs de comparaison pris en charge sont les suivants :

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  Voici la description de chaque opérateur de comparaison.

  - `EQ` : égal. L'opérateur `EQ` est pris en charge pour tous les types de données, y compris les listes et les mappages.

    `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre, Binaire, Ensemble de chaînes, Ensemble de nombres ou Ensemble de binaires. Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.
  - `NE` : pas égal. L'opérateur `NE` est pris en charge pour tous les types de données, y compris les listes et les mappages.

    `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre, Binaire, Ensemble de chaînes, Ensemble de nombres ou Ensemble de binaires. Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la

demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.

- `LE` : inférieur ou égal.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.

- `LT` : inférieur.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.

- `GE` : supérieur ou égal.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.

- `GT` : supérieur.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : l'attribut existe. L'opérateur `NOT_NULL` est pris en charge pour tous les types de données, y compris les listes et les mappages.

  > **Note**
  >
  > Cet opérateur teste l'existence d'un attribut, pas son type de données. Si le type de données de l'attribut « a » est nul, et si vous l'évaluez à l'aide de l'opérateur `NOT_NULL`,

le résultat est un booléen de valeur `true`. Ce résultat est dû au fait que l'attribut « a » existe. Son type de données n'est pas pertinent pour l'opérateur de comparaison `NOT_NULL`.

- `NULL` : l'attribut n'existe pas. L'opérateur `NULL` est pris en charge pour tous les types de données, y compris les listes et les mappages.

  > **Note**
  >
  > Cet opérateur teste la non-existence d'un attribut, pas son type de données. Si le type de données de l'attribut « a » est nul, et si vous l'évaluez à l'aide de l'opérateur `NULL`, le résultat est un booléen de valeur `false`. Cela est dû au fait que l'attribut « a » existe. Son type de données n'est pas pertinent pour l'opérateur de comparaison `NULL`.

- `CONTAINS` : vérifie la présence d'une sous-séquence ou d'une valeur dans un ensemble.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si l'attribut cible de la comparaison est de type Chaîne, l'opérateur vérifie la présence d'une sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type Binaire, l'opérateur vérifie la présence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est un ensemble (« `SS` », « `NS` » ou « `BS` »), l'opérateur évalue true s'il trouve une correspondance exacte avec un membre de l'ensemble.

  L'opérateur CONTAINS est pris en charge pour les listes. Lors de l'évaluation de « a `CONTAINS` b », « a » peut être une liste. En revanche, « b » ne peut pas être un ensemble, un mappage ou une liste.

- `NOT_CONTAINS` : vérifie l'absence d'une sous-séquence ou d'une valeur dans un ensemble.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si l'attribut cible de la comparaison est de type Chaîne, l'opérateur vérifie l'absence de sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type Binaire, l'opérateur vérifie l'absence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est un ensemble (« `SS` », « `NS` » ou « `BS` »), l'opérateur évalue true s'il ne trouve pas `does not` de correspondance exacte avec un membre de l'ensemble.

L'opérateur NOT_CONTAINS est pris en charge pour les listes. Lors de l'évaluation de « a NOT CONTAINS b », « a » peut être une liste. En revanche, « b » ne peut pas être un ensemble, un mappage ou une liste.

- BEGINS_WITH : vérifie la présence d'un préfixe.

  AttributeValueList ne peut contenir qu'un seul élément AttributeValue de type Chaîne ou Binaire (pas Nombre ou Ensemble). L'attribut cible de la comparaison doit être de type Chaîne ou Binaire (pas Nombre ou Ensemble).

- IN : vérifie la présence d'éléments correspondants dans deux ensembles.

  AttributeValueList peut contenir un ou plusieurs éléments AttributeValue de type Chaîne, Nombre ou Binaire (pas Ensemble). Ces attributs sont comparés à un attribut de type d'ensemble existant d'un élément. Si des éléments de l'ensemble en entrée sont présents dans l'attribut de l'élément, l'expression prend la valeur true.

- BETWEEN : supérieur ou égal à la première valeur, et inférieur ou égal à la deuxième.

  AttributeValueList doit contenir deux éléments AttributeValue du même type, Chaîne, Nombre ou Binaire (pas Ensemble). Un attribut cible correspond si la valeur cible est supérieure ou égale au premier élément, et inférieure ou égale au deuxième. Si une liste contient un élément AttributeValue d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, {"S":"6"} ne se compare pas à {"N":"6"}. De même, {"N":"6"} ne se compare pas à {"NS":["6", "2", "1"]}.

Les paramètres suivants peuvent être utilisés au lieu de AttributeValueList et ComparisonOperator :

- Value - Valeur que DynamoDB doit comparer avec un attribut.
- Exists - Valeur booléenne qui amène DynamoDB à évaluer la valeur avant de tenter l'opération conditionnelle :
  - Si Exists a la valeur true, DynamoDB vérifie si cette valeur d'attribut existe déjà dans la table. Si elle y figure, la condition a la valeur true. Sinon, la condition a la valeur false.
  - Si Exists a la valeur false, DynamoDB suppose que la valeur d'attribut n'existe not dans la table. Si la valeur n'existe effectivement pas, l'hypothèse est valide et le résultat de la condition est true. Si la valeur est trouvée malgré l'hypothèse qu'elle n'existe pas, le résultat de la condition est false.

Notez que la valeur par défaut pour `Exists` est `true`.

Les paramètres `Value` et `Exists` sont incompatibles avec `AttributeValueList` et `ComparisonOperator`. Notez que, si vous utilisez les deux ensembles de paramètres à la fois, DynamoDB renvoie une exception `ValidationException`.

> **Note**
>
> Ce paramètre ne prend pas en charge les attributs de type Liste ou Mappage.

Utiliser ConditionExpression à la place – Exemple

Supposons que vous souhaitiez modifier un élément dans la table Music, mais seulement si le résultat d'une certaine condition est true. Vous pourriez utiliser une requête `UpdateItem` avec un paramètre `Expected` comme dans cet exemple d'AWS CLI :

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

Vous pouvez utiliser `ConditionExpression` à la place :

```
aws dynamodb update-item \
    --table-name Music \
```

```
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

KeyConditions (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, utilisez KeyConditionExpression à la place..

Le paramètre conditionnel hérité `KeyConditions` contient des critères de sélection pour une opération `Query`. Pour une requête sur une table, vous ne pouvez avoir de conditions que sur les attributs de clé primaire de table. Vous devez fournir le nom et la valeur de la clé de partition comme condition `EQ`. Vous pouvez également fournir une deuxième condition pour la clé de tri.

> **Note**
>
> Si vous ne fournissez pas de condition de clé de tri, tous les éléments correspondant à la clé de partition sont extraits. Si les paramètres `FilterExpression` ou un `QueryFilter` sont présents, ils sont appliqués une fois les éléments extraits.

Pour une requête sur un index, vous ne pouvez avoir de conditions que sur les attributs de clé d'index. Vous devez fournir le nom et la valeur de la clé de partition d'index comme condition `EQ`. Vous pouvez également fournir une deuxième condition pour la clé de tri d'index.

Chaque élément `KeyConditions` se compose d'un nom d'attribut à comparer, ainsi que de ce qui suit :

- `AttributeValueList` - Une ou plusieurs valeurs à évaluer par rapport à l'attribut fourni. Le nombre de valeurs dans la liste dépend de l'opérateur `ComparisonOperator` utilisé.

  Pour le type Nombre, les comparaisons de valeurs sont numériques.

  Les comparaisons de valeurs de chaîne pour supérieur, égal ou inférieur sont basées sur le codage Unicode binaire UTF-8. Par exemple, a est supérieur à `A`, et a est supérieur à `B`.

  Pour le type Binaire, DynamoDB traite chaque octet des données binaires comme non signé lors de la comparaison de valeurs binaires.

- `ComparisonOperator` - Comparateur pour l'évaluation d'attributs. Par exemple : égal à, supérieur à, et inférieur à.

  Pour `KeyConditions`, seuls les opérateurs de comparaison suivants sont pris en charge :

  ```
  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN
  ```

  Voici une description de ces opérateurs de comparaison.

  - `EQ` : égal.

    `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas égal à `{"NS":["6", "2", "1"]}`.

  - `LE` : inférieur ou égal.

    `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.

  - `LT` : inférieur.

    `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple,

`{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.

- `GE` : supérieur ou égal.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` ne se compare pas à `{"NS":["6", "2", "1"]}`.

- `GT` : supérieur.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne, Nombre ou Binaire (pas Ensemble). Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas égal à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.

- `BEGINS_WITH` : vérifie la présence d'un préfixe.

  `AttributeValueList` ne peut contenir qu'un seul élément `AttributeValue` de type Chaîne ou Binaire (pas Nombre ou Ensemble). L'attribut cible de la comparaison doit être de type Chaîne ou Binaire (pas Nombre ou Ensemble).

- `BETWEEN` : supérieur ou égal à la première valeur, et inférieur ou égal à la deuxième.

  `AttributeValueList` doit contenir deux éléments `AttributeValue` du même type, Chaîne, Nombre ou Binaire (pas Ensemble). Un attribut cible correspond si la valeur cible est supérieure ou égale au premier élément, et inférieure ou égale au deuxième. Si une liste contient un élément `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, `{"S":"6"}` n'est pas comparable à `{"N":"6"}`. De même, `{"N":"6"}` n'est pas comparable à `{"NS":["6", "2", "1"]}`.

Utiliser KeyConditionExpression à la place – Exemple

Supposons que vous souhaitiez extraire plusieurs éléments avec la même clé de partition à partir de la table Music. Vous pourriez utiliser une requête `Query` avec un paramètre `KeyConditions` comme dans cet exemple d'AWS CLI :

```
aws dynamodb query \
```

```
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
    }'
```

Vous pouvez utiliser `KeyConditionExpression` à la place :

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

QueryFilter (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, utilisez FilterExpression à la place..

Dans une opération `Query`, le paramètre conditionnel hérité `QueryFilter` est une condition qui évalue les résultats de la requête après lecture des éléments et renvoie uniquement les valeurs souhaitées.

Ce paramètre ne prend pas en charge les attributs de type Liste ou Mappage.

> **Note**
>
> Un `QueryFilter` est appliqué après lecture des éléments. Le processus de filtrage ne consomme pas d'unités de capacité de lecture supplémentaires.

Si vous spécifiez plusieurs conditions dans le mappage `QueryFilter`, par défaut, le résultat de toutes les conditions doit être true. En d'autres termes, les conditions sont combinées à l'aide de l'opérateur `AND`. Vous pouvez également utiliser le paramètre ConditionalOperator (hérité) pour dissocier les conditions à l'aide de l'opérateur logique OR (OU). Dans ce cas, au moins une des conditions doit avoir la valeur vrai, plutôt que toutes.

Notez que `QueryFilter` n'autorise pas les attributs de clé. Vous ne pouvez pas définir de condition de filtre sur une clé de partition ou une clé de tri.

Chaque élément `QueryFilter` se compose d'un nom d'attribut à comparer, ainsi que de ce qui suit :

- `AttributeValueList` - Une ou plusieurs valeurs à évaluer par rapport à l'attribut fourni. Le nombre de valeurs dans la liste dépend de l'opérateur spécifié dans `ComparisonOperator`.

  Pour le type Nombre, les comparaisons de valeurs sont numériques.

  Les comparaisons de valeurs de chaîne pour supérieur, égal ou inférieur sont basées sur le codage binaire UTF-8. Par exemple, a est supérieur à `A`, et a est supérieur à `B`.

  Pour le type Binaire, DynamoDB traite chaque octet des données binaires comme non signé lors de la comparaison de valeurs binaires.

  Pour plus d'informations sur la spécification des types de données en JSON, consultez API de bas niveau de DynamoDB.
- `ComparisonOperator` - Comparateur pour l'évaluation d'attributs. Par exemple : égal à, supérieur à, et inférieur à.

  Les opérateurs de comparaison pris en charge sont les suivants :

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Utiliser FilterExpression à la place – Exemple

Supposons que vous vouliez interroger la table Music et appliquer une condition aux éléments correspondants. Vous pourriez utiliser une requête `Query` avec un paramètre `QueryFilter` comme dans cet exemple d'AWS CLI :

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
    }'
```

Vous pouvez utiliser `FilterExpression` à la place :

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

ScanFilter (hérité)

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB. Pour obtenir des informations précises sur le nouveau paramètre qui remplace celui-ci, utilisez FilterExpression à la place..

Dans une opération `Scan`, le paramètre conditionnel hérité `ScanFilter` est une condition qui évalue les résultats de l'analyse et renvoie uniquement les valeurs souhaitées.

> **Note**
>
> Ce paramètre ne prend pas en charge les attributs de type Liste ou Mappage.

Si vous spécifiez plusieurs conditions dans le mappage `ScanFilter`, par défaut, le résultat de toutes les conditions doit être true. En d'autres termes, les conditions sont associées à l'aide de l'opérateur logique AND (ET). Vous pouvez également utiliser le paramètre ConditionalOperator (hérité) pour dissocier les conditions à l'aide de l'opérateur logique OR (OU). Dans ce cas, au moins une des conditions doit avoir la valeur vrai, plutôt que toutes.

Chaque élément `ScanFilter` se compose d'un nom d'attribut à comparer, ainsi que de ce qui suit :

- `AttributeValueList` - Une ou plusieurs valeurs à évaluer par rapport à l'attribut fourni. Le nombre de valeurs dans la liste dépend de l'opérateur spécifié dans `ComparisonOperator`.

  Pour le type Nombre, les comparaisons de valeurs sont numériques.

  Les comparaisons de valeurs de chaîne pour supérieur, égal ou inférieur sont basées sur le codage binaire UTF-8. Par exemple, `a` est supérieur à `A`, et `a` est supérieur à `B`.

  Pour le type Binaire, DynamoDB traite chaque octet des données binaires comme non signé lors de la comparaison de valeurs binaires.

  Pour plus d'informations sur la spécification des types de données en JSON, consultez API de bas niveau de DynamoDB.
- `ComparisonOperator` - Comparateur pour l'évaluation d'attributs. Par exemple : égal à, supérieur à, et inférieur à.

  Les opérateurs de comparaison pris en charge sont les suivants :

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Utiliser FilterExpression à la place – Exemple

Supposons que vous vouliez analyser la table Music et appliquer une condition aux éléments correspondants. Vous pourriez utiliser une requête `Scan` avec un paramètre `ScanFilter` comme dans cet exemple d'AWS CLI :

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Vous pouvez utiliser `FilterExpression` à la place :

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Conditions d'écriture avec des paramètres hérités

> **Note**
>
> Dans la mesure du possible, nous vous recommandons d'utiliser les nouveaux paramètres d'expression plutôt que ces paramètres hérités. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB.

La section suivante décrit comment écrire des conditions pour une utilisation avec des paramètres hérités, tels que `Expected`, `QueryFilter` et `ScanFilter`.

> **Note**
>
> Les nouvelles applications doivent utiliser des paramètres d'expression à la place. Pour de plus amples informations, consultez Utilisation d'expressions dans DynamoDB.

Conditions simples

Avec des valeurs d'attribut, vous pouvez écrire des conditions pour des comparaisons par rapport à des attributs de table. Une condition produit toujours un résultat true (vrai) ou false (faux), et comprend les éléments suivants :

- `ComparisonOperator` – Supérieur, inférieur, égal, etc.
- `AttributeValueList` (facultatif) – Valeur(s) d'attribut à comparer. En fonction de l'opérateur `ComparisonOperator` utilisé, la liste `AttributeValueList` peut contenir une ou plusieurs valeurs ou être totalement absente.

Les sections suivantes décrivent les différents opérateurs de comparaison, et fournissent des exemples d'utilisation de ceux-ci dans des conditions.

Opérateurs de comparaison sans valeur d'attribut

- `NOT_NULL` – true si un attribut existe.
- `NULL` – true si aucun attribut n'existe.

Utilisez ces opérateurs pour vérifier si un attribut existe ou non. À défaut de valeur à comparer, ne spécifiez pas `AttributeValueList`.

Exemple

L'expression suivante a la valeur true si l'attribut Dimensions existe.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Opérateurs de comparaison avec une seule valeur d'attribut

- `EQ` – true si un attribut est égal à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre, Binaire, Ensemble de chaînes, Ensemble de nombres ou Ensemble de binaires. Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas. Par exemple, la chaîne `"3"` n'est pas égale au nombre `3`. En outre, le nombre `3` n'est pas égal à l'ensemble de nombres `[3, 2, 1]`.

- `NE` – true si un attribut n'est pas égal à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre, Binaire, Ensemble de chaînes, Ensemble de nombres ou Ensemble de binaires. Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

- `LE` – true si un attribut est inférieur ou égal à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si un élément contient une `AttributeValue` d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

- `LT` – true si un attribut est inférieur à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

- `GE` – true si un attribut est supérieur ou égal à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

- `GT` – true si un attribut est supérieur à une valeur.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

- `CONTAINS` – true si une valeur est présente dans un ensemble, ou si une valeur en contient une autre.

`AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si l'attribut cible de la comparaison est de type Chaîne, l'opérateur vérifie la présence d'une sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type Binaire, l'opérateur vérifie la présence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est un ensemble, l'opérateur évalue true s'il trouve une correspondance exacte avec un membre de l'ensemble.

- `NOT_CONTAINS` – true si une valeur n'est pas présente dans un ensemble, ou si une valeur n'en contient pas une autre.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne, Nombre ou Binaire (pas Ensemble). Si l'attribut cible de la comparaison est de type Chaîne, l'opérateur vérifie l'absence de sous-chaîne correspondante. Si l'attribut cible de la comparaison est de type Binaire, l'opérateur vérifie l'absence d'une sous-séquence de la cible correspondant à l'entrée. Si l'attribut cible de la comparaison est un ensemble, l'opérateur évalue true s'il ne trouve pas une correspondance exacte avec un membre de l'ensemble.

- `BEGINS_WITH` – true si les premiers caractères d'un attribut correspondent à la valeur fournie. N'utilisez pas cet opérateur pour comparer des nombres.

  `AttributeValueList` ne peut contenir qu'une seule valeur de type Chaîne ou Binaire (pas Nombre ou Ensemble). L'attribut cible de la comparaison doit être une chaîne ou un binaire (pas un nombre ou un ensemble).

Utilisez ces opérateurs pour comparer un attribut avec une valeur. Vous devez spécifier une `AttributeValueList` contenant une valeur unique. Pour la plupart des opérateurs, cette valeur doit être un scalaire, mais les opérateurs `EQ` et `NE` prennent également en charge des ensembles.

Exemples

Les expressions suivantes prennent la valeur true si :

- Le prix d'un produit est supérieur à 100.

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
```

```
...
```

- Une catégorie de produits commence par « Bo ».

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- Un produit est disponible en rouge, vert ou noir :

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Lors de la comparaison de types de données d'ensemble, l'ordre des éléments n'a pas d'importance. DynamoDB renvoie uniquement les éléments ayant le même ensemble de valeurs, quel que soit l'ordre dans lequel vous les spécifiez dans votre demande.

Opérateurs de comparaison avec deux valeurs d'attribut

- `BETWEEN` – true si une valeur s'inscrit entre une limite inférieure et une limite supérieure, points de terminaison inclus.

  `AttributeValueList` doit contenir deux éléments du même type, Chaîne, Nombre ou Binaire (pas Ensemble). Un attribut cible correspond si la valeur cible est supérieure ou égale au premier élément, et inférieure ou égale au deuxième. Si un élément contient une valeur d'un type différent de celui spécifié dans la demande, la valeur ne correspond pas.

Utilisez cet opérateur pour déterminer si une valeur d'attribut s'inscrit dans une plage. `AttributeValueList` doit contenir deux éléments scalaires du même type, Chaîne, Nombre ou Binaire.

Exemple

L'expression suivante prend la valeur true si le prix d'un produit est compris entre 100 et 200.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Opérateurs de comparaison avec n valeurs d'attribut

- `IN` – true si une valeur est égale à l'une des valeurs d'une liste énumérée. Seules les valeurs scalaires sont prises en charge dans la liste, pas les ensembles. Pour correspondre, l'attribut cible doit être du même type et avoir exactement la même valeur.

  `AttributeValueList` peut contenir un ou plusieurs éléments de type Chaîne, Nombre ou Binaire (pas Ensemble). Ces attributs sont comparés à un attribut de type autre qu'Ensemble existant d'un élément. Si des éléments de l'ensemble en entrée sont présents dans l'attribut de l'élément, l'expression prend la valeur true.

  `AttributeValueList` peut contenir une ou plusieurs valeurs de type Chaîne, Nombre ou Binaire (pas Ensemble). Pour correspondre, l'attribut cible de la comparaison doit être du même type et avoir exactement la même valeur. Une chaîne ne correspond jamais à un ensemble de chaîne.

Utilisez cet opérateur pour déterminer si la valeur fournie figure dans une liste énumérée. Vous pouvez spécifier n'importe quel nombre de valeurs scalaires dans `AttributeValueList`, mais elles doivent toutes être du même type de données.

Exemple

L'expression suivante prend la valeur true si la valeur d'Id est 201, 203 ou 205.

```
...
    "Id": {
```

```
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Utilisation de plusieurs conditions

DynamoDB permet de combiner plusieurs conditions pour former des expressions complexes. Pour cela, fournissez au moins deux expressions avec un opérateur ConditionalOperator (hérité) facultatif.

Par défaut, lorsque vous spécifiez plusieurs conditions, toutes les conditions doivent avoir la valeur true pour que l'expression entière prenne la valeur true. En d'autres termes, une opération AND implicite a lieu.

Exemple

L'expression suivante prend la valeur true si un produit est un livre qui contient au moins 600 pages. Les résultats des deux conditions doivent être true, car elles sont implicitement associées avec l'opérateur logique AND.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

Vous pouvez utiliser l'opérateur ConditionalOperator (hérité) pour préciser qu'une opération AND va avoir lieu. L'exemple suivant se comporte de la même manière que le précédent.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
```

```
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

Vous pouvez également définir l'opérateur `ConditionalOperator` sur OR, ce qui signifie que le résultat d'au moins une des conditions doit être true.

Exemple

L'expression suivante prend la valeur true si un produit est un VTT (mountain bike), s'il est d'une marque particulière ou si son prix est supérieur à 100.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> Dans une expression complexe, les conditions sont traitées dans l'ordre, de la première à la dernière.
>
> Vous ne pouvez pas utiliser les opérateurs logiques AND et OR dans une même expression.

Autres opérateurs conditionnels

Dans les versions précédentes de DynamoDB, le paramètre `Expected` se comportait différemment pour les écritures conditionnelles. Chaque élément du mappage `Expected` représentait un nom d'attribut à vérifier par DynamoDB, ainsi que les éléments suivants :

- `Value` – Valeur à laquelle comparer l'attribut.
- `Exists` – Détermine si la valeur existe avant de tenter l'opération.

Les options `Value` et `Exists` sont toujours prises en charge dans DynamoDB. Toutefois, elles vous permettent uniquement de tester une condition d'égalité ou l'existence d'un attribut. Nous vous recommandons d'utiliser `ComparisonOperator` et `AttributeValueList` à la place, car ces options vous permettent de construire un éventail beaucoup plus large de conditions.

Example

Une opération `DeleteItem` peut vérifier si un livre n'est plus publié, et ne le supprimer que si le résultat de cette condition est true. Voici un exemple AWS CLI utilisant une condition héritée :

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

L'exemple suivant fait la même chose, mais n'utilise pas de condition héritée :

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Une opération `PutItem` peut protéger contre le remplacement d'un élément existant par les mêmes attributs de clé primaire. Voici un exemple utilisant une condition héritée :

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

L'exemple suivant fait la même chose, mais n'utilise pas de condition héritée :

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Pour les conditions dans le mappage `Expected`, n'utilisez pas les options `Value` et `Exists` héritées avec `ComparisonOperator` et `AttributeValueList`. Si vous le faites, votre écriture conditionnelle échouera.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.