aws

Guida per l'utente

Amazon Aurora DSQL

Amazon Aurora DSQL: Guida per l’utente

Table of Contents

Che cos'è Amazon Aurora DSQL?

Amazon Aurora DSQL è un servizio di database relazionale distribuito e serverless ottimizzato per i carichi di lavoro transazionali. Aurora DSQL offre una scalabilità praticamente illimitata e non richiede la gestione dell'infrastruttura. L'architettura active-active ad alta disponibilità offre una disponibilità del 99,99% in un'unica Regione e del 99,999% in più Regioni.

Quando utilizzare Aurora DSQL

Aurora DSQL è ottimizzato per carichi di lavoro transazionali che traggono vantaggio dalle transazioni ACID e da un modello di dati relazionale. Poiché è serverless, Aurora DSQL è ideale per modelli applicativi di architetture basate su microservizi, serverless e basate su eventi. Aurora DSQL è compatibile con PostgreSQL, quindi puoi usare driver familiari, mappature relazionali a oggetti (), framework e funzionalità SQL. ORMs

Aurora DSQL gestisce automaticamente l'infrastruttura di sistema e dimensiona l'elaborazione, l'I/O e lo storage in base al carico di lavoro. Poiché non sono presenti server di cui fare il provisioning o da gestire, non è necessario preoccuparsi del tempo di inattività per la manutenzione legati al provisioning, all'applicazione di patch o agli aggiornamenti dell'infrastruttura.

Aurora DSQL permette di creare e mantenere applicazioni aziendali sempre disponibili su qualsiasi scala. Il design serverless active-active automatizza il ripristino degli errori, quindi non è necessario preoccuparsi del tradizionale failover del database. Le applicazioni traggono vantaggio dalla disponibilità multi-AZ e multi-Regione e non bisogna preoccuparsi dell'eventuale coerenza o della mancanza di dati relativi ai failover.

Funzionalità principali di Aurora DSQL

Le seguenti funzionalità principali consentono di creare un database distribuito serverless per supportare le applicazioni ad alta disponibilità:

Architettura distribuita

Aurora è costituito dai seguenti componenti multi-tenant:

- Relay e connettività
- Calcolo e database

- Log delle transazioni, controllo della concorrenza e isolamento
- Storage

Un piano di controllo (control-plane) coordina i componenti precedenti. Ogni componente fornisce ridondanza su tre zone di disponibilità (), con ridimensionamento automatico del cluster e riparazione automatica in caso di guasti dei componenti. AZs Per maggiori informazioni su come questa architettura supporta l'alta disponibilità, consulta Resilienza in Amazon Aurora DSQL.

Cluster a Regione singola e multi-Regione

I cluster Aurora DSQL offrono i seguenti vantaggi:

- Replica sincrona dei dati
- Operazioni di lettura consistenti
- Ripristino automatico dei guasti
- Coerenza dei dati tra più o più regioni AZs

In caso di guasto di un componente dell'infrastruttura, Aurora DSQL indirizza automaticamente le richieste verso un'infrastruttura funzionante senza intervento manuale. Aurora DSQL fornisce transazioni con caratteristiche ACID (atomicità, coerenza, isolamento e durabilità) con una forte coerenza, isolamento degli snapshot, atomicità e durabilità inter-AZ e inter-regionale.

I cluster in peering multi-Regione offrono la stessa resilienza e connettività dei cluster a Regione singola. Tuttavia, migliorano la disponibilità offrendo due endpoint regionali, uno in ogni Regione di cluster in peering. Entrambi gli endpoint di un cluster in peering presentano un unico database logico. Sono disponibili per operazioni di lettura e scrittura simultanee e forniscono una forte coerenza dei dati. È possibile creare applicazioni che vengono eseguite in più Regioni contemporaneamente per garantire prestazioni e resilienza, con la certezza che i lettori vedano sempre gli stessi dati.

Compatibilità con i database PostgreSQL

Il livello di database distribuito (calcolo) in Aurora DSQL si basa su una versione principale corrente di PostgreSQL. È possibile connettersi ad Aurora DSQL con driver e strumenti PostgreSQL comuni, come `psql`. Aurora DSQL è attualmente compatibile con PostgreSQL versione 16 e supporta un sottoinsieme di funzionalità, espressioni e tipi di dati di PostgreSQL. Per maggiori informazioni sulle funzionalità supportate, consulta Compatibilità delle funzionalità SQL in Aurora DSQL.

Disponibilità regionale per Aurora DSQL

Con Amazon Aurora DSQL, puoi distribuire istanze di database su più istanze Regioni AWS per supportare applicazioni globali e soddisfare i requisiti di residenza dei dati. La disponibilità della Regione determina dove è possibile creare e gestire i cluster di database Aurora DSQL. Gli amministratori di database e gli architetti delle applicazioni che devono progettare sistemi di database ad alta disponibilità e distribuiti a livello globale spesso devono comprendere il supporto regionale per i propri carichi di lavoro. I casi d'uso più comuni includono la configurazione del disaster recovery tra Regioni, l'assistenza agli utenti da istanze di database geograficamente più vicine per ridurre la latenza e la conservazione delle copie dei dati in posizioni specifiche per motivi di conformità.

La tabella seguente mostra Regioni AWS dove Aurora DSQL è attualmente disponibile e l'endpoint per ciascuno di essi. Regione AWS

Nome della regione	Regione	Endpoint	Protocollo	
US East (Ohio)	us-east-2	dsql.us-east-2.api.aws dsql-fips.us-east-2.api.aws	HTTPS HTTPS	
US East (N. Virginia)	us-east-1	dsql.us-east-1.api.aws dsql-fips.us-east-1.api.aws	HTTPS HTTPS	
Stati Uniti occidentali (Oregon)	us-west-2	dsql.us-west-2.api.aws dsql-fips.us-west-2.api.aws	HTTPS HTTPS	
Asia Pacifico (Osaka-Locale)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS	

Nome della regione	Regione	Endpoint	Protocollo	
Asia Pacifico (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS	
Asia Pacifico (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS	
Europa (Francoforte)	eu-central-1	dsql.eu-central-1.api.aws	HTTPS	
Europa (Irlanda)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS	
Europa (Londra)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS	
Europa (Parigi)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS	

Disponibilità di cluster multi-Regione per Aurora DSQL

È possibile creare cluster multiregione Aurora DSQL all'interno di set di regioni specifici. AWS Ogni set di Regioni raggruppa Regioni geograficamente correlate che possono lavorare insieme in un cluster multi-Regione.

Regioni degli Stati Uniti

- Stati Uniti orientali (Virginia settentrionale)
- Stati Uniti orientali (Ohio)
- Stati Uniti occidentali (Oregon)

Regioni dell'Asia Pacifico

- Asia Pacifico (Osaka-Locale)
- Asia Pacific (Seul)
- Asia Pacifico (Tokyo)

Regioni europee

- Europa (Francoforte)
- Europa (Irlanda)
- Europe (London)
- Europa (Parigi)

Limitazioni importanti

I cluster multi-Regione devono essere creati all'interno di un singolo set di Regioni. Ad esempio, non è possibile creare un cluster che includa le Regioni Stati Uniti orientali (Virginia settentrionale) ed Europa (Irlanda).

> **Important**
>
> Aurora DSQL al momento non supporta i cluster multi-Regione su molteplici continenti.

Prezzi di Aurora DSQL

Per informazioni sui costi, consulta Prezzi di Aurora DSQL.

Fasi successive

Per informazioni sui componenti principali di Aurora DSQL e per iniziare a utilizzare il servizio, consulta quanto segue:

- Nozioni di base su Aurora DSQL
- Compatibilità delle funzionalità SQL in Aurora DSQL
- Accesso ad Aurora DSQL con client compatibili con PostgreSQL

- Aurora DSQL e PostgreSQL

Nozioni di base su Aurora DSQL

Amazon Aurora DSQL è un database relazionale distribuito e serverless ottimizzato per carichi di lavoro transazionali. Nelle sezioni seguenti, imparerai come creare cluster Aurora DSQL a regione singola e multiarea, connetterti a essi e creare e caricare uno schema di esempio. Accederai ai cluster con la AWS Console e opzionalmente interagirai con il tuo database utilizzando altri client PostgreSQL. Alla fine, avrai un cluster Aurora DSQL funzionante pronto all'uso per carichi di lavoro di test o produzione.

Argomenti

Prerequisiti

Prima di cominciare, assicurarsi che i seguenti requisiti preliminari siano soddisfatti:

- La tua identità IAM deve disporre dell'autorizzazione per accedere alla console.
- La tua identità IAM deve soddisfare i seguenti criteri:
  - Accesso per eseguire qualsiasi azione su qualsiasi risorsa del tuo Account AWS
  - `AmazonAuroraDSQLConsoleFullAccess` AWS la politica gestita è allegata.

Passo 1: Creazione di cluster Aurora DSQL a Regione singola

L'unità base di Aurora DSQL è il cluster, che è il luogo in cui vengono archiviati i dati. In questa attività, viene creato un cluster in una singola Regione AWS.

Come creare un cluster a Regione singola in Aurora DSQL

1. Accedi a Console di gestione AWS e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql

2. Scegli Crea cluster e poi Regione singola.
3. (Facoltativo) modifica il valore del tag Name predefinito.
4. (Facoltativo) Aggiungi tag aggiuntivi per questo cluster.
5. (Facoltativo) Nelle Impostazioni cluster, seleziona una delle seguenti opzioni:
   - Seleziona Personalizza le impostazioni di crittografia (avanzate) per scegliere o creare una AWS KMS key.
   - Seleziona Abilita protezione da eliminazione per impedire che un'operazione possa eliminare il cluster. Per impostazione predefinita, la protezione dall'eliminazione è abilitata.
   - Seleziona Politica basata sulle risorse (avanzata) per specificare le politiche di controllo degli accessi per questo cluster.
6. Scegli Crea cluster.
7. La console riporta alla pagina Cluster. Viene visualizzato un banner di notifica che indica che il cluster è in fase di creazione. Seleziona l'ID del cluster per aprire la visualizzazione dei dettagli del cluster.

Passo 2: Connessione al cluster Aurora DSQL

Aurora DSQL supporta diversi modi per connettersi al cluster, tra cui DSQL Query Editor AWS CloudShell, il client psql locale e altri strumenti compatibili con PostgreSQL. In questo passaggio, ci si connette utilizzando Aurora DSQL Query Editor, che fornisce un modo rapido per iniziare a interagire con il nuovo cluster.

Per connettersi utilizzando il Query Editor

1. Nella console Aurora DSQL (https://console.aws.amazon.com/dsql), apri la pagina Cluster e conferma che la creazione del cluster è stata completata e che il relativo stato è Attivo.
2. Seleziona il cluster dall'elenco o scegli l'ID del cluster per aprire la pagina dei dettagli del cluster.
3. Scegli Connect with Query editor.
4. Scegli Connect as admin per il cluster appena creato.
   - Facoltativamente, puoi connetterti con un ruolo personalizzato, vedi Utilizzo dei ruoli del database e dell'autenticazione IAM.

Passo 3: Esecuzione di comandi SQL di esempio in Aurora DSQL

Effettua test sul cluster Aurora DSQL eseguendo istruzioni SQL. Dopo aver aperto il cluster nel Query Editor, seleziona ed esegui ogni query di esempio passo dopo passo.

Esegui comandi SQL di esempio in Aurora DSQL

1. Crea uno schema denominato `test`.

```
CREATE SCHEMA IF NOT EXISTS test;
```

2. Crea una tabella hello_world che utilizza un UUID generato automaticamente come chiave primaria.

```
CREATE TABLE IF NOT EXISTS test.hello_world (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    message VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Inserisci una riga di esempio.

```
INSERT INTO test.hello_world (message)
VALUES ('Hello, World!');
```

4. Leggete i valori inseriti.

```
SELECT * FROM test.hello_world;
```

5. Opzionalmente pulisci

```
DROP TABLE test.hello_world;
DROP SCHEMA test;
```

Fase 4 (opzionale): Creare un cluster multiregionale

Per creare un cluster multi-Regione, è necessario specificare le seguenti Regioni:

Regione remota

Questa è la Regione nella quale viene creato un secondo cluster. Si crea un secondo cluster in questa Regione e si esegue il peering al cluster iniziale. Aurora DSQL replica tutte le scritture verso il cluster iniziale sul cluster remoto. È possibile leggere e scrivere su qualsiasi cluster.

Regione testimone

Questa Regione riceve tutti i dati scritti nel cluster multi-Regione. Tuttavia, le Regioni testimone non ospitano gli endpoint per i client e non forniscono l'accesso ai dati degli utenti. Una finestra limitata del log delle transazioni crittografato viene mantenuta nelle Regioni testimone. Questo log facilita il ripristino e supporta il quorum transazionale nel caso in cui una Regione non risultasse più disponibile.

Utilizzare la procedura seguente per creare un cluster iniziale, creare un secondo cluster in una regione diversa e quindi eseguire il peering dei due cluster per creare un cluster multiregionale. Dimostra inoltre la replica delle scritture tra Regioni e le letture coerenti da entrambi gli endpoint regionali.

Creazione di un cluster multi-Regione

1. Accedi alla console Aurora DSQL.
2. Nel pannello di navigazione scegliere Cluster.
3. Scegli Crea cluster e poi multi-Regione.
4. (Facoltativo) modifica il valore del tag Name predefinito.
5. (Facoltativo) Aggiungi tag aggiuntivi per questo cluster.
6. In Impostazioni multi-Regione, scegli le seguenti opzioni per il cluster iniziale:
   - In Regione testimone, scegli una Regione. Attualmente, come Regioni testimone dei cluster multi-Regione sono supportate solo le Regioni localizzate negli Stati Uniti.
   - (Facoltativo) In ARN del cluster della Regione remota, inserisci un ARN per un cluster esistente in un'altra Regione. Se non esiste alcun cluster che funga da secondo cluster nel cluster multi-Regione, completa la configurazione dopo aver creato il cluster iniziale.
7. (Facoltativo) In Impostazioni cluster, seleziona una delle seguenti opzioni per il cluster iniziale:
   - Seleziona Personalizza le impostazioni di crittografia (avanzate) per scegliere o creare una AWS KMS key.

- Seleziona Abilita protezione da eliminazione per impedire che un'operazione possa eliminare il cluster. Per impostazione predefinita, la protezione dall'eliminazione è abilitata.
- Seleziona Politica basata sulle risorse (avanzata) per specificare le politiche di controllo degli accessi per questo cluster.

8. Per creare il cluster iniziale, scegli Crea cluster. Se non è stato inserito un ARN nel passaggio precedente, la console mostra la notifica Configurazione del cluster in sospeso.
9. Nella notifica Configurazione del cluster in sospeso, scegli Completa configurazione del cluster multi-Regione. Questa azione avvia la creazione di un secondo cluster in un'altra Regione.
10. Scegli una delle seguenti opzioni per il secondo cluster:

    - Aggiungi ARN del cluster della Regione remota: scegli questa opzione se esiste un cluster e desideri che sia il secondo cluster del cluster multi-Regione.
    - Crea cluster in un'altra Regione: scegli questa opzione per creare un secondo cluster. In Regione remota, scegli la Regione per questo secondo cluster.

11. Scegli Crea cluster in ***your-second-region***, *your-second-region* dov'è la posizione del secondo cluster. La console si apre sulla seconda Regione.
12. (Facoltativo) Scegli le impostazioni del cluster per il secondo cluster. Ad esempio, puoi scegliere una AWS KMS key.
13. Per creare il secondo cluster, scegli Crea cluster.
14. Scegli Peer in ***initial-cluster-region***, *initial-cluster-region* dov'è la regione che ospita il primo cluster che hai creato.
15. Quando richiesto, scegli Conferma. Questo passo completa la creazione del cluster multi-Regione.

Come connettersi al secondo cluster

1. Apri la console di Aurora DSQL e scegli la Regione per il secondo cluster.
2. Scegli Cluster.
3. Seleziona la riga per il secondo cluster del cluster multi-Regione.
4. Scegli Connect with Query editor.
5. Scegli Connettiti come amministratore.
6. Crea uno schema e una tabella di esempio e inserisci i dati seguendo i passaggi indicatiPasso 3: Esecuzione di comandi SQL di esempio in Aurora DSQL.

Come interrogare i dati nel secondo cluster dalla Regione che ospita il cluster iniziale

1. Nella console Aurora DSQL, scegli la Regione del cluster iniziale.
2. Scegliere Cluster.
3. Seleziona la riga per il secondo cluster del cluster multi-Regione.
4. Scegli Connect with Query editor.
5. Scegli Connettiti come amministratore.
6. Interrogare i dati inseriti nel secondo cluster.

   Example

   ```
   SELECT * FROM test.hello_world;
   ```

Risoluzione dei problemi

Consulta la sezione Risoluzione dei problemi della documentazione di Aurora DSQL.

Autenticazione e autorizzazione per Aurora DSQL

Aurora DSQL utilizza i ruoli e le policy IAM per l'autorizzazione dei cluster. Per l'autorizzazione del database si associano i ruoli IAM ai ruoli del database PostgreSQL. Questo approccio combina i vantaggi di IAM con i privilegi di PostgreSQL. Aurora DSQL utilizza queste funzionalità per fornire una policy di autorizzazione e accesso completa per cluster, database e dati.

Gestione del cluster tramite IAM

Per gestire il cluster, utilizza IAM per l'autenticazione e le autorizzazioni:

Autenticazione IAM

Per autenticare l'identità IAM quando si gestiscono i cluster Aurora DSQL è necessario utilizzare IAM. È possibile fornire l'autenticazione utilizzando Console di gestione AWS, AWS CLI o l'SDK AWS.

Autorizzazione IAM

Per gestire i cluster Aurora DSQL, concedi l'autorizzazione utilizzando le azioni IAM per Aurora DSQL. Ad esempio, per descrivere un cluster, assicurati che la tua identità IAM disponga delle autorizzazioni per l'azione IAM `dsql:GetCluster`, come nell'esempio seguente di azione della policy.

```
{
  "Effect": "Allow",
  "Action": "dsql:GetCluster",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Per maggiori informazioni, consulta Utilizzo delle azioni delle policy IAM per gestire i cluster.

Connessione al cluster tramite IAM

Per connetterti al cluster, utilizza IAM per l'autenticazione e le autorizzazioni:

Autenticazione IAM

Genera un token di autenticazione temporaneo utilizzando un'identità IAM con autorizzazione per connetterti al cluster. Per maggiori informazioni, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.

Autorizzazione IAM

Concedi le seguenti azioni della policy IAM all'identità IAM che stai utilizzando per stabilire la connessione all'endpoint del cluster:

- Utilizza `dsql:DbConnectAdmin` se stai utilizzando il ruolo `admin`. Aurora DSQL crea e gestisce questo ruolo per l'utente. Il seguente esempio di azione politica IAM consente `admin` di connettersi a*`my-cluster`*.

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnectAdmin",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

- Utilizza `dsql:DbConnect` se stai utilizzando un ruolo del database personalizzato. Puoi creare e gestire questo ruolo utilizzando i comandi SQL nel database. Il seguente esempio di azione politica IAM consente la connessione di un ruolo di database personalizzato *`my-cluster`* per un massimo di un'ora.

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnect",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Dopo aver stabilito una connessione, il ruolo è autorizzato per un massimo di un'ora.

Interazione con il database utilizzando i ruoli del database PostgreSQL e i ruoli IAM

PostgreSQL gestisce le autorizzazioni di accesso al database utilizzando il concetto di ruoli. Un ruolo può essere considerato come un utente del database o un gruppo di utenti del database, a seconda di come è impostato. I ruoli PostgreSQL vengono creati utilizzando i comandi SQL. Per gestire

l'autorizzazione a livello di database, concedi le autorizzazioni PostgreSQL ai ruoli del database PostgreSQL.

Aurora DSQL supporta due tipi di ruoli del database: un ruolo `admin` e i ruoli personalizzati. Aurora DSQL crea automaticamente un ruolo `admin` predefinito nel cluster Aurora DSQL. Non è possibile modificare il ruolo `admin`. Quando ci si connette al tuo database come `admin`, è possibile utilizzare comandi SQL per creare nuovi ruoli a livello di database da associare ai ruoli IAM. Per consentire ai ruoli IAM di connettersi al database, associare i ruoli del database personalizzati ai ruoli IAM.

Autenticazione

Utilizza il ruolo `admin` per connetterti al cluster. Dopo aver connesso il database, utilizza il comando `AWS IAM GRANT` per associare un ruolo del database personalizzato all'identità IAM autorizzata a connettersi al cluster, come nell'esempio seguente.

```
AWS IAM GRANT custom-db-role TO 'arn:aws:iam::account-id:role/iam-role-name';
```

Per maggiori informazioni, consultare Autorizzazione alla connessione al cluster per i ruoli del database.

Autorizzazione

Utilizza il ruolo `admin` per connetterti al cluster. Esegui i comandi SQL per configurare ruoli di database personalizzati e concedere le autorizzazioni. Per maggiori informazioni, consulta le pagine dedicate ai ruoli del database PostgreSQL e ai privilegi PostgreSQL nella documentazione di PostgreSQL.

Utilizzo delle azioni delle policy IAM con Aurora DSQL

L'azione della policy IAM utilizzata dipende dal ruolo utilizzato per la connessione al cluster: un ruolo `admin` o un ruolo del database personalizzato. La policy dipende anche dalle azioni IAM richieste per questo ruolo.

Utilizzo delle azioni delle policy IAM per connettersi ai cluster

Quando ti connetti al cluster con il ruolo del database predefinito di `admin`, utilizza un'identità IAM con autorizzazione per eseguire le seguenti azioni delle policy IAM.

```
"dsql:DbConnectAdmin"
```

Quando ti connetti al cluster con un ruolo del database personalizzato, associa innanzitutto il ruolo IAM al ruolo del database. L'identità IAM usata per connettersi al cluster deve essere autorizzata a eseguire le seguenti azioni delle policy IAM.

```
"dsql:DbConnect"
```

Per maggiori informazioni sui ruoli del database personalizzati, consulta Utilizzo dei ruoli del database e dell'autenticazione IAM.

Utilizzo delle azioni delle policy IAM per gestire i cluster

Quando gestisci i cluster Aurora DSQL, specifica le azioni delle policy solo per le azioni che il tuo ruolo deve eseguire. Ad esempio, se il ruolo deve solo ottenere informazioni sul cluster, è possibile limitare le autorizzazioni del ruolo alle sole autorizzazioni `GetCluster` e `ListClusters`, come illustrato nella seguente policy di esempio

JSON

```
{
"Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
    }
  ]
}
```

La seguente policy di esempio mostra tutte le azioni delle policy IAM disponibili per la gestione dei cluster.

JSON

```
{
```

```
"Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:UpdateCluster",
        "dsql:DeleteCluster",
        "dsql:ListClusters",
        "dsql:TagResource",
        "dsql:ListTagsForResource",
        "dsql:UntagResource"
      ],
      "Resource" : "*"
    }
  ]
}
```

Revoca dell'autorizzazione tramite IAM e PostgreSQL

È possibile revocare le autorizzazioni di accesso ai ruoli a livello di database dei ruoli IAM:

Revoca dell'autorizzazione dell'amministratore alla connessione ai cluster

Per revocare l'autorizzazione alla connessione al cluster con il ruolo `admin`, revoca l'accesso dell'identità IAM a `dsql:DbConnectAdmin`. Modifica la policy IAM o scollega la policy dall'identità.

Dopo aver revocato l'autorizzazione di connessione all'identità IAM, Aurora DSQL rifiuta tutti i nuovi tentativi di connessione da tale identità IAM. Qualsiasi connessione attiva che utilizza l'identità IAM potrebbe rimanere autorizzata per tutta la durata della connessione. Per maggiori informazioni sulle durate della connessione, consulta Quote e limiti.

Revoca dell'autorizzazione personalizzata al ruolo per la connessione ai cluster

Per revocare l'accesso a ruoli del database diversi da `admin`, revoca l'accesso dell'identità IAM a `dsql:DbConnect`. Modifica la policy IAM o scollega la policy dall'identità.

È anche possibile rimuovere l'associazione tra il ruolo del database e IAM utilizzando il comando `AWS IAM REVOKE` nel database. Per maggiori informazioni sulla revoca dell'accesso ai ruoli del database, consulta Revoca dell'autorizzazione del database a un ruolo IAM.

Non è possibile gestire le autorizzazioni del ruolo predefinito `admin` del database. Per informazioni su come gestire le autorizzazioni per i ruoli del database personalizzati, consulta Privilegi di PostgreSQL. Le modifiche ai privilegi hanno effetto sulla transazione successiva dopo che Aurora DSQL ha eseguito correttamente il commit della transazione di modifica.

Generazione di un token di autenticazione in Amazon Aurora DSQL

Per connetterti ad Amazon Aurora DSQL con un client SQL, genera un token di autenticazione da utilizzare come password. Questo token viene utilizzato solo per autenticare la connessione. Una volta stabilita la connessione, la connessione rimane valida anche se il token di autenticazione scade.

Se crei un token di autenticazione utilizzando la AWS console, per impostazione predefinita il token scade automaticamente dopo un'ora. Se si utilizza AWS CLI o SDKs per creare il token, l'impostazione predefinita è 15 minuti. La durata massima è di 604.800 secondi, ovvero una settimana. Per connetterti nuovamente ad Aurora DSQL dal client, puoi utilizzare lo stesso token di autenticazione se non è scaduto oppure è possibile generarne uno nuovo.

Per iniziare a generare un token, crea una policy IAM e un cluster in Aurora DSQL. Quindi usa la AWS console o AWS CLI il AWS SDKs per generare un token.

È necessario disporre almeno delle autorizzazioni IAM elencate in Connessione al cluster tramite IAM, a seconda del ruolo del database utilizzato per la connessione.

Argomenti

- Usa la AWS console per generare un token di autenticazione in Aurora DSQL
- AWS CloudShellDa utilizzare per generare un token di autenticazione in Aurora DSQL
- Usa il AWS CLI per generare un token di autenticazione in Aurora DSQL
- Usa il SDKs per generare un token in Aurora DSQL

Usa la AWS console per generare un token di autenticazione in Aurora DSQL

Aurora DSQL autentica gli utenti con un token anziché una password. Puoi generare il token dalla console.

Per generare un token di autenticazione

1. Accedi a Console di gestione AWS e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. Seleziona l'ID del cluster per cui desideri generare un token di autenticazione. Se non è ancora stato creato un cluster, segui i passaggi indicati in Passo 1: Creazione di cluster Aurora DSQL a Regione singola o Fase 4 (opzionale): Creare un cluster multiregionale.
3. Scegli Connetti, quindi seleziona Ottieni token.
4. Scegli se vuoi connetterti come `admin` o con un ruolo del database personalizzato.
5. Copia il token di autenticazione generato e utilizzalo per Accedi ad Aurora DSQL utilizzando client SQL.

Per maggiori informazioni sui ruoli del database personalizzati e su IAM in Aurora DSQL, consulta Autenticazione e autorizzazione.

AWS CloudShellDa utilizzare per generare un token di autenticazione in Aurora DSQL

Prima di poter generare un token di autenticazione utilizzandoAWS CloudShell, assicurati di creare un cluster Aurora DSQL.

Per generare un token di autenticazione utilizzando AWS CloudShell

1. Accedi a Console di gestione AWS e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. In basso a sinistra della AWS console, scegli. AWS CloudShell
3. Esegui il comando seguente per generare un token di autenticazione per il ruolo `admin`. Sostituisci *`us-east-1`* con la tua regione e *`your_cluster_endpoint`* con l'endpoint del tuo cluster.

> **Note**
>
> Se non ti connetti come `admin`, utilizza invece `generate-db-connect-auth-token`.

```
aws dsql generate-db-connect-admin-auth-token \
```

```
  --expires-in 3600 \
  --region us-east-1 \
  --hostname your_cluster_endpoint
```

In caso di problemi, consulta Risoluzione dei problemi di IAM e Come posso risolvere gli errori di accesso negato o di operazione non autorizzata con una policy IAM?

4. Utilizza il comando seguente per utilizzare `psql` per aprire una connessione al cluster.

```
PGSSLMODE=require \
psql --dbname postgres \
  --username admin \
  --host cluster_endpoint
```

5. Dovrebbe comparire un prompt di immissione della password. Copia il token che generato e assicurati di non includere spazi o caratteri aggiuntivi. Incollalo nel seguente prompt da `psql`.

```
Password for user admin:
```

6. Premere Invio. Dovrebbe comparire un prompt di PostgreSQL.

```
postgres=>
```

Se ricevi un errore di accesso negato, assicurati che la tua identità IAM disponga dell'autorizzazione `dsql:DbConnectAdmin`. Se disponi dell'autorizzazione e continui a ricevere errori di accesso negato, consulta Risoluzione dei problemi di IAM e Come posso risolvere gli errori di accesso negato o di operazione non autorizzata con una policy IAM?

Per maggiori informazioni sui ruoli del database personalizzati e su IAM in Aurora DSQL, consulta Autenticazione e autorizzazione.

Usa il AWS CLI per generare un token di autenticazione in Aurora DSQL

Se il cluster è in stato `ACTIVE`, è possibile generare un token di autenticazione dalla CLI utilizzando il comando `aws dsql`. È possibile utilizzare una delle seguenti tecniche:

- Se ti stai connettendo con il ruolo `admin`, utilizza l'opzione `generate-db-connect-admin-auth-token`.

- Se ti stai connettendo con un ruolo del database personalizzato, utilizza l'opzione `generate-db-connect-auth-token`.

L'esempio seguente utilizza i seguenti attributi per generare un token di autenticazione per il ruolo `admin`.

- *`your_cluster_endpoint`*— L'endpoint del cluster. Segue il formato *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, come nell'esempio `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*— llRegione AWS, ad esempio `us-east-2` o`us-east-1`.

Gli esempi seguenti impostano il tempo di scadenza del token in 3.600 secondi (1 ora).

Linux and macOS

```
aws dsql generate-db-connect-admin-auth-token \
  --region region \
  --expires-in 3600 \
  --hostname your_cluster_endpoint
```

Windows

```
aws dsql generate-db-connect-admin-auth-token ^
  --region=region ^
  --expires-in=3600 ^
  --hostname=your_cluster_endpoint
```

Usa il SDKs per generare un token in Aurora DSQL

È possibile generare un token di autenticazione per il cluster quando è in stato `ACTIVE`. Gli esempi basati su SDK utilizzano i seguenti attributi per generare un token di autenticazione per il ruolo `admin`:

- *`your_cluster_endpoint`*(o*`yourClusterEndpoint`*) — L'endpoint del cluster Aurora DSQL. Il formato di denominazione è *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, come nell'esempio `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.

- *region*(o*RegionEndpoint*) — L'area Regione AWS in cui si trova il cluster, ad esempio o. `us-east-2` `us-east-1`

Python SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `generate_db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `generate_connect_auth_token`.

```
def generate_token(your_cluster_endpoint, region):
    client = boto3.client("dsql", region_name=region)
    # use `generate_db_connect_auth_token` instead if you are not connecting as
 admin.
    token = client.generate_db_connect_admin_auth_token(your_cluster_endpoint,
 region)
    print(token)
    return token
```

C++ SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `GenerateDBConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `GenerateDBConnectAuthToken`.

```
#include <aws/core/Aws.h>
#include <aws/dsql/DSQLClient.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;

std::string generateToken(String yourClusterEndpoint, String region) {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQLClient client{clientConfig};
    std::string token = "";

    // If you are not using the admin role to connect, use
 GenerateDBConnectAuthToken instead
    const auto presignedString =
 client.GenerateDBConnectAdminAuthToken(yourClusterEndpoint, region);
    if (presignedString.IsSuccess()) {
        token = presignedString.GetResult();
    } else {
        std::cerr << "Token generation failed." << std::endl;
    }

    std::cout << token << std::endl;

    Aws::ShutdownAPI(options);
    return token;
}
```

JavaScript SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `getDbConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `getDbConnectAuthToken`.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";

async function generateToken(yourClusterEndpoint, region) {
  const signer = new DsqlSigner({
    hostname: yourClusterEndpoint,
    region,
  });
  try {
    // Use `getDbConnectAuthToken` if you are _not_ logging in as the `admin` user
    const token = await signer.getDbConnectAdminAuthToken();
    console.log(token);
    return token;
  } catch (error) {
      console.error("Failed to generate token: ", error);
      throw error;
  }
}
```

Java SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `generateDbConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `generateDbConnectAuthToken`.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.dsql.DsqlUtilities;
import software.amazon.awssdk.regions.Region;

public class GenerateAuthToken {
    public static String generateToken(String yourClusterEndpoint, Region region) {
        DsqlUtilities utilities = DsqlUtilities.builder()
                .region(region)
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Use `generateDbConnectAuthToken` if you are _not_ logging in as `admin`
 user
        String token = utilities.generateDbConnectAdminAuthToken(builder -> {
            builder.hostname(yourClusterEndpoint)
                    .region(region);
        });

        System.out.println(token);
        return token;
    }
}
```

Rust SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `db_connect_auth_token`.

```
use aws_config::{BehaviorVersion, Region};
use aws_sdk_dsql::auth_token::{AuthTokenGenerator, Config};

async fn generate_token(your_cluster_endpoint: String, region: String) -> String {
    let sdk_config = aws_config::load_defaults(BehaviorVersion::latest()).await;
    let signer = AuthTokenGenerator::new(
        Config::builder()
            .hostname(&your_cluster_endpoint)
            .region(Region::new(region))
            .build()
            .unwrap(),
    );

    // Use `db_connect_auth_token` if you are _not_ logging in as `admin` user
    let token = signer.db_connect_admin_auth_token(&sdk_config).await.unwrap();
    println!("{}", token);
    token.to_string()
}
```

Ruby SDK

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `generate_db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `generate_db_connect_auth_token`.

```
require 'aws-sdk-dsql'

def generate_token(your_cluster_endpoint, region)
  credentials = Aws::SharedCredentials.new()

  begin
      token_generator = Aws::DSQL::AuthTokenGenerator.new({
          :credentials => credentials
      })

      # if you're not using admin role, use generate_db_connect_auth_token instead
      token = token_generator.generate_db_connect_admin_auth_token({
          :endpoint => your_cluster_endpoint,
```

```
          :region => region
      })
  rescue => error
    puts error.full_message
  end
end
```

.NET

> **Note**
>
> L'SDK ufficiale per .NET non include una chiamata API integrata per generare un token di autenticazione per Aurora DSQL. È necessario utilizzare invece `DSQLAuthTokenGenerator`, che è una classe di utilità. Nell'esempio di codice riportato di seguito viene illustrato come generare il token di autenticazione in .NET.

È possibile generare il token nei modi seguenti:

- Se ti stai connettendo con il ruolo `admin`, utilizza `DbConnectAdmin`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza `DbConnect`.

L'esempio seguente utilizza la classe di utilità `DSQLAuthTokenGenerator` per generare il token di autenticazione per un utente con il ruolo `admin`. *`insert-dsql-cluster-endpoint`*Sostituiscilo con l'endpoint del cluster.

```
using Amazon;
using Amazon.DSQL.Util;
using Amazon.Runtime;

var yourClusterEndpoint = "insert-dsql-cluster-endpoint";

AWSCredentials credentials = FallbackCredentialsFactory.GetCredentials();

var token = DSQLAuthTokenGenerator.GenerateDbConnectAdminAuthToken(credentials,
 RegionEndpoint.USEast1, yourClusterEndpoint);

Console.WriteLine(token);
```

Golang

> **Note**
>
> L'SDK Golang non fornisce un metodo integrato per generare un token prefirmato. È necessario creare manualmente la richiesta firmata, come illustrato nel seguente esempio di codice.

Nel seguente esempio di codice, specifica il valore `action` sulla base dell'utente PostgreSQL:

- Se ti stai connettendo con il ruolo `admin`, utilizza l'azione `DbConnectAdmin`.
- Se ti stai connettendo con un ruolo del database personalizzato, utilizza l'azione `DbConnect`.

Oltre a *`yourClusterEndpoint`* e*`region`*, l'esempio seguente utilizza*`action`*. Specificare il in *`action`* base all'utente PostgreSQL.

```
func GenerateDbConnectAdminAuthToken(yourClusterEndpoint string, region
 string, action string) (string, error) {
 // Fetch credentials
 sess, err := session.NewSession()
 if err != nil {
  return "", err
 }

 creds, err := sess.Config.Credentials.Get()
 if err != nil {
  return "", err
 }
 staticCredentials := credentials.NewStaticCredentials(
  creds.AccessKeyID,
  creds.SecretAccessKey,
  creds.SessionToken,
 )

 // The scheme is arbitrary and is only needed because validation of the URL
 requires one.
 endpoint := "https://" + yourClusterEndpoint
 req, err := http.NewRequest("GET", endpoint, nil)
 if err != nil {
  return "", err
 }
 values := req.URL.Query()
 values.Set("Action", action)
 req.URL.RawQuery = values.Encode()

 signer := v4.Signer{
  Credentials: staticCredentials,
 }
 _, err = signer.Presign(req, nil, "dsql", region, 15*time.Minute, time.Now())
 if err != nil {
  return "", err
 }

 url := req.URL.String()[len("https://"):]

 return url, nil
}
```

Utilizzo dei ruoli del database e dell'autenticazione IAM

Aurora DSQL supporta l'autenticazione utilizzando sia i ruoli IAM che gli utenti IAM. È possibile utilizzare entrambi i metodi per autenticare e accedere ai database Aurora DSQL.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche ma non è associata a una persona specifica. Utilizzo di ruoli IAM per fornire credenziali di sicurezza temporanee. È possibile assumere temporaneamente un ruolo IAM in modi diversi:

- Cambiando ruolo nel Console di gestione AWS
- Richiamando un'operazione AWS CLI o AWS API
- Utilizzando un URL personalizzato

Dopo aver assunto un ruolo, puoi accedere ad Aurora DSQL utilizzando le credenziali temporanee del ruolo. Per maggiori informazioni sui metodi per l'utilizzo dei ruoli, consulta Identità IAM nella Guida per l'utente di IAM.

Utenti IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche ed è associata a una singola persona o applicazione. Gli utenti IAM dispongono di credenziali a lungo termine come password e chiavi di accesso che possono essere utilizzate per accedere ad Aurora DSQL.

> **Note**
>
> Per eseguire comandi SQL con l'autenticazione IAM, puoi utilizzare il ruolo IAM ARNs o l'utente IAM ARNs negli esempi seguenti.

Autorizzazione alla connessione al cluster per i ruoli del database

Creazione di un ruolo IAM e concessione dell'autorizzazione alla connessione con l'azione della policy IAM: `dsql:DbConnect`.

La policy IAM deve inoltre concedere l'autorizzazione ad accedere alle risorse del cluster. Utilizza una wildcard (*) o segui le istruzioni riportate nella pagina che spiega come utilizzare le chiavi di condizione IAM con Amazon Aurora DSQL.

Autorizzazione dei ruoli del database a utilizzare SQL nel database

È necessario utilizzare un ruolo IAM con autorizzazione a connettersi al cluster.

1. Connettiti al cluster Aurora DSQL utilizzando un'utilità SQL.

   Utilizza il ruolo del database `admin` con un'identità IAM autorizzata all'azione IAM `dsql:DbConnectAdmin` per connetterti al cluster.
2. Crea un nuovo ruolo nel database, assicurandoti di specificare l'opzione `WITH LOGIN`.

   ```
   CREATE ROLE example WITH LOGIN;
   ```
3. Associa il ruolo del database all'ARN del ruolo IAM.

   ```
   AWS IAM GRANT example TO 'arn:aws:iam::012345678912:role/example';
   ```
4. Concedi autorizzazioni a livello di database al ruolo del database

   Negli esempi seguenti viene utilizzato il comando `GRANT` per fornire l'autorizzazione all'interno del database.

   ```
   GRANT USAGE ON SCHEMA myschema TO example;
   GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA myschema TO example;
   ```

Per maggiori informazioni, consulta PostgreSQL `GRANT` e Privilegi di PostgreSQL nella documentazione di PostgreSQL.

Visualizzazione delle mappature tra IAM e i ruoli del database

Per visualizzare le mappature tra i ruoli IAM e i ruoli del database, interroga la tabella di sistema `sys.iam_pg_role_mappings`.

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Output di esempio:

```
 iam_oid |                  arn                   | pg_role_oid | pg_role_name |
 grantor_pg_role_oid | grantor_pg_role_name
---------+----------------------------------------+-------------+--------------
+---------------------+----------------------
   26398 | arn:aws:iam::012345678912:role/example |       26396 | example      |
        15579 | admin
(1 row)
```

Questa tabella mostra tutte le mappature tra i ruoli IAM (identificati dal relativo ARN) e i ruoli del database PostgreSQL.

Revoca dell'autorizzazione del database a un ruolo IAM

Per revocare l'autorizzazione del database, utilizzare l'operazione `AWS IAM REVOKE`.

```
AWS IAM REVOKE example FROM 'arn:aws:iam::012345678912:role/example';
```

Per maggiori informazioni sulla revoca dell'autorizzazione, consulta Revoca dell'autorizzazione tramite IAM e PostgreSQL.

Aurora DSQL e PostgreSQL

Aurora DSQL è un database relazionale distribuito compatibile con PostgreSQL progettato per carichi di lavoro transazionali. Aurora DSQL utilizza componenti di base di PostgreSQL come parser, planner, optimizer e type system.

Il design di Aurora DSQL garantisce che tutta la sintassi PostgreSQL supportata fornisca un comportamento compatibile e produca risultati di query identici. Ad esempio, Aurora DSQL fornisce conversioni di tipo, operazioni aritmetiche e precisione e scalabilità numeriche identiche a PostgreSQL. Eventuali deviazioni sono documentate.

Aurora DSQL introduce anche funzionalità avanzate come il controllo ottimistico della concorrenza e la gestione distribuita dello schema. Con queste funzionalità, è possibile utilizzare gli strumenti familiari di PostgreSQL beneficiando al contempo delle prestazioni e della scalabilità richieste per applicazioni distribuite moderne e native del cloud.

Aspetti salienti della compatibilità con PostgreSQL

Aurora DSQL è attualmente basato sulla versione 16 di PostgreSQL. Le compatibilità principali includono le seguenti:

Protocollo Wire

Aurora DSQL utilizza il protocollo wire standard PostgreSQL v3. Ciò consente l'integrazione con client, driver e strumenti PostgreSQL standard. Ad esempio, Aurora DSQL è compatibile con `psql`, `pgjdbc` e `psycopg`.

Compatibilità SQL

Aurora DSQL supporta un'ampia gamma di espressioni e funzioni PostgreSQL standard comunemente utilizzate nei carichi di lavoro transazionali. Le espressioni SQL supportate producono risultati identici a PostgreSQL, tra cui:

- Gestione dei valori nulli
- Comportamento dell'ordinamento
- Scala e precisione per le operazioni numeriche
- Equivalenza per le operazioni sulle stringhe

Per maggiori informazioni, consultare Compatibilità delle funzionalità SQL in Aurora DSQL.

Gestione delle transazioni

Aurora DSQL conserva le caratteristiche principali di PostgreSQL, come le transazioni ACID e un livello di isolamento equivalente a PostgreSQL Repeatable Read. Per maggiori informazioni, consultare Controllo della concorrenza in Aurora DSQL.

Differenze chiave dell'architettura

Il design distribuito e senza condivisione di Aurora DSQL presenta alcune differenze fondamentali rispetto a PostgreSQL tradizionale. Queste differenze sono parte integrante dell'architettura Aurora DSQL e offrono molti vantaggi in termini di prestazioni e scalabilità. Le differenze principali includono le seguenti:

Controllo ottimistico della concorrenza (OCC)

Aurora DSQL utilizza un modello ottimistico di controllo della concorrenza. Questo approccio senza blocchi impedisce alle transazioni di bloccarsi a vicenda, elimina i deadlock e consente l'esecuzione parallela ad alto throughput. Queste funzionalità rendono Aurora DSQL particolarmente utile per le applicazioni che richiedono prestazioni costanti su larga scala. Per ulteriori esempi, consulta Controllo della concorrenza in Aurora DSQL.

Operazioni DDL asincrone

Aurora DSQL esegue le operazioni DDL in modo asincrono, il che consente letture e scritture ininterrotte durante le modifiche allo schema. La sua architettura distribuita consente ad Aurora DSQL di eseguire le seguenti operazioni:

- Esecuzione di operazioni DDL come attività in background, riducendo al minimo le interruzioni.
- Coordinamento delle modifiche al catalogo come transazioni distribuite fortemente coerenti. Ciò garantisce la visibilità atomica su tutti i nodi, anche in caso di malfunzionamenti o operazioni simultanee.
- Funzionamento in modo completamente distribuito e senza leader su più zone di disponibilità con livelli di elaborazione e storage disaccoppiati.

Per ulteriori informazioni sull'utilizzo del comando EXPLAIN in PostgreSQL, vedere. DDL e transazioni distribuite in Aurora DSQL

Compatibilità delle funzionalità SQL in Aurora DSQL

Aurora DSQL e PostgreSQL restituiscono risultati identici per tutte le query SQL. Si noti che Aurora DSQL si differenzia da PostgreSQL senza una clausola `ORDER BY`. Nelle sezioni seguenti sono disponibile ulteriori informazioni sul supporto di Aurora DSQL per i tipi di dati e i comandi SQL di PostgreSQL.

Argomenti

Tipi di dati supportati in Aurora DSQL

Aurora DSQL supporta un sottoinsieme di questi tipi di dati comuni di PostgreSQL.

Argomenti

Tipi di dati numerici

Aurora DSQL supporta i seguenti tipi di dati numerici di PostgreSQL.

Name	Alias	Intervallo e precisione	Dimensioni dell'archiviazione	Supporto dell'indicizzazione
`smallint`	`int2`	Da -32768 a +32767	2 byte	Sì

Name	Alias	Intervallo e precisione	Dimensioni dell'archiviazione	Supporto dell'indicizzazione
`integer`	`int`, `int4`	Da -2147483648 a +2147483647	4 byte	Sì
`bigint`	`int8`	Da -9223372036854775808 a 9223372036854775807	8 byte	Sì
`real`	`float4`	Precisione a 6 cifre decimali	4 byte	Sì
`double precision`	`float8`	Precisione a 15 cifre decimali	8 byte	Sì
`numeric [ (p, s) ]`	`decimal [ (p, s) ]` `dec[ (p,s)]`	Numerico esatto con precisione selezionabile. La precisione massima è 38 e la scala massima è 37.[1] Il valore predefinito è `numeric (18,6)`.	8 byte + 2 byte per cifra di precisione. La dimensione massima è di 27 byte.	No

[1] - Se non si specifica esplicitamente una dimensione durante l'esecuzione di `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL applica le impostazioni predefinite. Aurora DSQL applica dei limiti durante l'esecuzione delle istruzioni `INSERT` o `UPDATE`.

Tipi di dati dei caratteri

Aurora DSQL supporta i seguenti tipi di dati di caratteri PostgreSQL.

Name	Alias	Description	Limite di Aurora SQL	Dimensioni dell'archiviazione	Supporto dell'indicizzazione
`character [ (n) ]`	`char [ (n) ]`	Stringa di caratteri a lunghezza fissa	4096 byte[1]	Variabile fino a 4100 byte	Sì

Name	Alias	Description	Limite di Aurora SQL	Dimensioni dell'archiviazione	Supporto dell'indicizzazione
`character varying [ (n) ]`	`varchar [ (n) ]`	Stringa di caratteri a lunghezza variabile	65535 byte[1]	Variabile fino a 65539 byte	Sì
`bpchar [ (n) ]`		Se a lunghezza fissa, si tratta di un alias per `char`. Se a lunghezza variabile, si tratta di un alias per `varchar`, dove gli spazi finali sono semanticamente insignificanti.	4096 byte[1]	Variabile fino a 4100 byte	Sì
`text`		Stringa di caratteri a lunghezza variabile	1 MiB[1]	Variabile fino a 1 MiB	Sì

[1] - Se non si specifica esplicitamente una dimensione quando si eseguono le istruzioni `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL applica le impostazioni predefinite. Aurora DSQL applica dei limiti durante l'esecuzione delle istruzioni `INSERT` o `UPDATE`.

Tipi di dati data e ora

Aurora DSQL supporta i seguenti tipi di dati di data e ora PostgreSQL.

Name	Alias	Description	Intervallo	Risoluzione	Dimensi dell'archiviazion	Supporto dell'indicizzazione
`date`		Data di calendario (anno, mese, giorno)	4713 A.C. - 5874897 D.C.	1 giorno	4 byte	Sì
`time[(p)][without time zone]`	`time`	Ora del giorno senza fuso orario	0 – 1	1 microsecondo	8 byte	Sì
`time[(p)] with time zone`	`timet`	ora del giorno, compreso il fuso orario	00:00:00+1559 – 24:00:00 –1559	1 microsecondo	12 byte	No
`timestamp [(p)][without time zone]`		Data e ora, senza fuso orario	4713 A.C. - 294276 D.C.	1 microsecondo	8 byte	Sì
`timestamp[(p)]with time zone`	`timetz`	Data e ora, incluso il fuso orario	4713 A.C. - 294276 D.C.	1 microsecondo	8 byte	Sì
`interval[fields][(p)]`		Intervallo di tempo	-178000000 anni - 178000000 anni	1 microsecondo	16 byte	No

Tipi di dati vari

Aurora DSQL supporta i seguenti tipi di dati PostgreSQL vari.

Name	Alias	Description	Limite di Aurora SQL	Dimensioni dell'archiviazione	Supporto dell'indicizzazione
`boolean`	`bool`	Booleano logico (true/false)		1 byte	Sì
`bytea`		Dati binari ("array di byte")	1 MiB[1]	Variabile fino al limite di 1 MiB	No
`UUID`		Identificatore univoco universale		16 byte	Sì

[1] - Se non si specifica esplicitamente una dimensione quando si eseguono le istruzioni `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL applica le impostazioni predefinite. Aurora DSQL applica dei limiti durante l'esecuzione delle istruzioni `INSERT` o `UPDATE`.

Tipi di dati di runtime delle query

I tipi di dati di runtime delle query sono tipi di dati interni utilizzati al momento dell'esecuzione delle query. Questi tipi sono distinti dai tipi compatibili con PostgreSQL come `varchar` e `integer` che è possibile definire nello schema. Questi tipi sono invece rappresentazioni di runtime utilizzate da Aurora DSQL per l'elaborazione di una query.

I seguenti tipi di dati sono supportati solo durante il runtime delle query:

Tipo array

Aurora DSQL supporta array dei tipi di dati supportati. Ad esempio, è possibile avere un array di numeri interi. La funzione `string_to_array` divide una stringa in un array in stile PostgreSQL con il delimitatore virgola (`,`) come mostrato nell'esempio seguente. È possibile utilizzare gli array nelle espressioni, negli output di funzioni o nei calcoli temporanei durante l'esecuzione delle query.

```
SELECT string_to_array('1,2', ',');
```

La funzione restituisce un risultato simile al seguente:

```
 string_to_array
-----------------
 {1,2}
(1 row)
```

Tipo inet

Il tipo di dati rappresenta IPv4 gli indirizzi IPv6 host e le relative sottoreti. Questo tipo è utile per l'analisi dei log, il filtraggio su sottoreti IP o l'esecuzione di calcoli di rete all'interno di una query. Per maggiori informazioni, consulta inet nella documentazione di PostgreSQL.

Funzioni JSON di runtime

Aurora DSQL supporta JSON e JSONB come tipi di dati di runtime per l'elaborazione delle query, sebbene questi non possano essere utilizzati come tipi di dati di colonna negli schemi di tabelle. È possibile archiviare dati JSON come `text` e trasformarli in JSON durante l'esecuzione delle query per utilizzare le funzioni e gli operatori JSON di PostgreSQL.

Aurora DSQL supporta la maggior parte delle funzioni JSON di PostgreSQL della sezione 9.1.6 Funzioni e operatori JSON con un comportamento identico. Le funzioni non supportate includono le funzioni aggregate: `json_agg`, `json_agg_strict`, `json_arrayagg`, `json_objectagg`, `json_object_agg`, `json_object_agg_strict`, `json_object_agg_unique`.

Le funzioni che restituiscono tipi JSON o JSONB potrebbero richiedere una trasformazione aggiuntiva in `text` per una corretta visualizzazione.

```
SELECT json_build_array(1, 2, 'foo', 4, 5)::text;
```

La funzione restituisce un risultato simile al seguente:

```
    json_build_array
---------------------
  [1, 2, "foo", 4, 5]
(1 row)
```

SQL supportato per Aurora DSQL

Aurora DSQL supporta un'ampia serie di funzionalità SQL di base di PostgreSQL. Nelle sezioni seguenti, è possibile ottenere informazioni sul supporto generale delle espressioni PostgreSQL. Questo elenco non è completo.

Comando **SELECT**

Aurora DSQL supporta le seguenti clausole del comando `SELECT`.

Clausola primaria	Clausole supportate
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	
`WITH` (espressioni di tabella comuni)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`

Clausola primaria	Clausole supportate
OVER	RANK (), PARTITION BY
FOR UPDATE	

Data Definition Language (DDL)

Aurora DSQL supporta i seguenti comandi DDL di PostgreSQL.

Comando	Clausola primaria	Clausole supportate
CREATE	TABLE	Per informazioni sulla sintassi supportata del comando CREATE TABLE, consulta CREATE TABLE.
ALTER	TABLE	Per informazioni sulla sintassi supportata del comando ALTER TABLE, consulta ALTER TABLE.
DROP	TABLE	
CREATE	[UNIQUE] INDEX ASYNC	È possibile eseguire questo comando con i seguenti parametri: ON, NULLS FIRST, NULLS LAST. Per informazioni sulla sintassi supportata del comando CREATE INDEX ASYNC, consulta Indici asincroni in Aurora SQL.
DROP	INDEX	
CREATE	VIEW	Per maggiori informazioni sulla sintassi supportata del comando CREATE VIEW, consulta CREATE VIEW.

Comando	Clausola primaria	Clausole supportate
ALTER	VIEW	Per informazioni sulla sintassi supportat a del comando ALTER VIEW, consulta ALTER VIEW.
DROP	VIEW	Per informazioni sulla sintassi supportata del comando DROP VIEW, consulta DROP VIEW.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Data Manipulation Language (DML)

Aurora DSQL supporta i seguenti comandi DML di PostgreSQL.

Comando	Clausola primaria	Clausole supportate
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Data Control Language (DCL)

Aurora DSQL supporta i seguenti comandi DCL di PostgreSQL.

Comando	Clausole supportate
GRANT	ON, TO

Comando	Clausole supportate
`REVOKE`	`ON`, `FROM`, `CASCADE`, `RESTRICT`

Transaction Control Language (TCL)

Aurora DSQL supporta i seguenti comandi TCL di PostgreSQL.

Comando	Clausole supportate
`COMMIT`	

`BEGIN`	`[WORK | TRANSACTION ]`
	
	`[READ ONLY | READ WRITE]`

Comandi di utilità

Aurora DSQL supporta i seguenti comandi di utilità di PostgreSQL:

- `EXPLAIN`
- `ANALYZE` (solo nome della relazione)

Sottoinsiemi di comandi SQL supportati in Aurora DSQL

Questa sezione di PostgreSQL fornisce informazioni dettagliate sulle espressioni supportate, concentrandosi sui comandi con set di parametri e sottocomandi estesi. Ad esempio, CREATE TABLE in PostgreSQL offre molte clausole e parametri. Questa sezione descrive tutti gli elementi della sintassi PostgreSQL supportati da Aurora DSQL per questi comandi.

Argomenti

CREATE TABLE

CREATE TABLE definisce una nuova tabella.

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression )|
  DEFAULT default_expr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |

and like_option is:

{ INCLUDING | EXCLUDING } { COMMENTS | CONSTRAINTS | DEFAULTS | GENERATED | IDENTITY |
 INDEXES | STATISTICS | ALL }

index_parameters in UNIQUE, and PRIMARY KEY constraints are:
[ INCLUDE ( column_name [, ... ] ) ]
```

ALTER TABLE

ALTER TABLE modifica la definizione di una tabella.

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
```

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema

where action is one of:

    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
```

CREATE VIEW

`CREATE VIEW` definisce una nuova vista persistente. Aurora DSQL non supporta le viste temporanee, sono supportate solo le viste permanenti.

Sintassi supportata

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Description

`CREATE VIEW` definisce una vista basata su una query. La vista non è materializzata fisicamente. Al contrario, la query viene eseguita ogni volta che si fa riferimento alla vista in una query.

`CREATE or REPLACE VIEW` è simile, ma se esiste già una vista con lo stesso nome, questa viene sostituita. La nuova query deve generare le stesse colonne generate dalla query della vista esistente (ovvero gli stessi nomi di colonna nello stesso ordine e con gli stessi tipi di dati), ma può aggiungere ulteriori colonne alla fine dell'elenco. I calcoli che danno origine alle colonne di output possono essere diversi.

Se viene specificato un nome di schema (come `CREATE VIEW myschema.myview ...`), la vista viene creata utilizzando lo schema specificato. In caso contrario la vista viene creata nello schema corrente.

Il nome della vista deve essere distinto dal nome di qualsiasi altra relazione (tabella, indice, vista) nello stesso schema.

Parameters

`CREATE VIEW` supporta vari parametri per controllare il comportamento delle viste aggiornabili automaticamente.

`RECURSIVE`

Crea una vista ricorsiva. La sintassi: `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` è equivalente a `CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;`.

Per una vista ricorsiva è necessario specificare un elenco di nomi di colonne della vista.

`name`

Il nome della vista da creare, che può essere facoltativamente accompagnato dallo schema. Per una vista ricorsiva è necessario specificare un elenco di nomi di colonne.

`column_name`

Un elenco facoltativo di nomi da utilizzare per le colonne nella vista. Se non vengono specificati nomi di colonne, questi ricavati dalla query.

`WITH ( view_option_name [= view_option_value] [, ... ] )`

Questa clausola specifica i parametri opzionali per una vista. Sono supportati i seguenti parametri.

- `check_option (enum)` - Questo parametro può assumere i valori `local` o `cascaded` ed è equivalente a specificare `WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)` - Deve essere usato se la vista è destinata a fornire una sicurezza a livello di riga. Aurora DSQL attualmente non supporta la sicurezza a livello di riga, ma questa opzione forzerà comunque la valutazione prioritaria delle condizioni `WHERE` della vista (e di tutte le condizioni che utilizzano operatori contrassegnati come `LEAKPROOF`).
- `security_invoker (boolean)` - Questa opzione fa sì che le relazioni di base sottostanti vengano verificate con i privilegi dell'utente della vista anziché con quelli del proprietario della vista. Per dettagli completi, consulta le note riportate di seguito.

Tutte le opzioni precedenti possono essere modificate nelle viste esistenti utilizzando `ALTER VIEW`.

`query`

Un comando `SELECT` o `VALUES` che fornisce le colonne e le righe della vista.

- `WITH [ CASCADED | LOCAL ] CHECK OPTION` - Questa opzione controlla il comportamento delle viste aggiornabili automaticamente. Quando viene specificata questa opzione, i comandi `INSERT` e `UPDATE` sulla vista verranno controllati per garantire che le nuove righe soddisfino la condizione di definizione della vista (ovvero, le nuove righe vengono controllate per garantire che siano visibili attraverso la vista). In caso contrario, l'aggiornamento verrà rifiutato. Se `CHECK OPTION` non è specificato, i comandi `INSERT` e `UPDATE` sulla vista possono creare righe che non sono visibili attraverso la vista stessa. Di seguito sono riportate le opzioni supportate.
- `LOCAL` - Le nuove righe vengono verificate solo in base alle condizioni definite direttamente nella vista stessa. Qualsiasi condizione definita nelle viste di base sottostanti non viene verificata (a meno che anch'esse non specifichino l'opzione `CHECK OPTION`).
- `CASCADED` - Le nuove righe vengono verificate rispetto alle condizioni della vista e di tutte le viste di base sottostanti. Se viene indicato `CHECK OPTION` e non viene specifica l'opzioni `LOCAL` né l'opzione `CASCADED`, viene assunto il valore `CASCADED`.

> **Note**
>
> `CHECK OPTION` può essere utilizzato con le viste `RECURSIVE`. `CHECK OPTION` è supportato solo nelle viste aggiornabili automaticamente.

Note

Utilizzare l'istruzione `DROP VIEW` per eliminare le viste.

I nomi e i tipi di dati delle colonne della vista devono essere considerati attentamente. Ad esempio, CREATE VIEW vista AS SELECT 'Hello World'; non è consigliato perché il nome della colonna predefinito è `?column?;`. Inoltre, il tipo di dati della colonna predefinito è `text`, il che potrebbe non essere quello desiderato.

Un approccio migliore consiste nello specificare esplicitamente il nome della colonna e il tipo di dati, ad esempio: `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

Per impostazione predefinita, l'accesso alle relazioni di base sottostanti a cui si fa riferimento nella vista è determinato dalle autorizzazioni del proprietario della vista. In alcuni casi, questo può essere

utilizzato per fornire un accesso sicuro ma limitato alle tabelle sottostanti. Tuttavia, non tutte le viste sono protette dalla manomissione.

- Se la proprietà `security_invoker` della vista è impostata su true, l'accesso alle relazioni di base sottostanti è determinato dalle autorizzazioni dell'utente che esegue la query, anziché su quelle del proprietario della vista. Pertanto, l'utente di una vista con l'opzione Security Invoker deve disporre delle autorizzazioni pertinenti sulla vista e sulle relative relazioni di base sottostanti.
- Se una delle relazioni di base sottostanti è una vista con l'opzione Security Invoker, verrà trattata come se vi fosse stato effettuato l'accesso direttamente dalla query originale. Pertanto, una vista con l'opzione Security Invoker verificherà sempre le relazioni di base sottostanti utilizzando le autorizzazioni dell'utente corrente, anche se vi si accede da una vista senza la proprietà `security_invoker`.
- Le funzioni chiamate nella vista vengono trattate come se fossero state chiamate direttamente dalla query che utilizza la vista. Pertanto, l'utente di una vista deve disporre delle autorizzazioni per chiamare tutte le funzioni utilizzate dalla vista. Le funzioni nella vista vengono eseguite con i privilegi dell'utente che esegue la query o del proprietario della funzione, a seconda che le funzioni siano definite come `SECURITY INVOKER` o `SECURITY DEFINER`. Ad esempio, la chiamata `CURRENT_USER` diretta in una vista restituirà sempre l'utente che la invoca, non il proprietario della vista. Ciò non è influenzato dall'impostazione dell'opzione `security_invoker` della vista, quindi una vista con l'opzione `security_invoker` impostata su false non è equivalente a una funzione `SECURITY DEFINER`.
- L'utente che crea o sostituisce una vista deve disporre dei privilegi `USAGE` su tutti gli schemi a cui si fa riferimento nella query della vista, al fine di poter accedere agli oggetti a cui si fa riferimento in tali schemi. Si noti, tuttavia, che questa ricerca viene eseguita solo quando la vista viene creata o sostituita. Pertanto, l'utente della vista richiede il privilegio `USAGE` solo sullo schema che contiene la vista, non sugli schemi a cui si fa riferimento nella query della vista, anche per una vista con l'opzione Security Invoker.
- Quando `CREATE OR REPLACE VIEW` viene utilizzato su una vista esistente, vengono modificate solo la regola di `SELECT` di definizione della vista, più eventuali parametri `WITH ( ... )` e relativa `CHECK OPTION`. Le altre caratteristiche della vista, tra cui proprietà, autorizzazioni e regole non selezionate, rimangono invariate. Per sostituire una vista è necessario esserne proprietari (ciò include essere un membro del ruolo proprietario).

Viste aggiornabili

Le viste semplici sono aggiornabili automaticamente: il sistema consentirà alle istruzioni `INSERT`, `UPDATE` e `DELETE` di operare sulla vista allo stesso modo in cui avrebbe fatto su una normale tabella. Una vista è aggiornabile automaticamente se soddisfa tutte le seguenti condizioni:

- La vista deve avere esattamente una voce nell'elenco della clausola `FROM`, e tale elemento deve essere una tabella o un'altra vista aggiornabile.
- La definizione della vista non deve contenere clausole `WITH`, `DISTINCT`, `GROUP BY`, `HAVING`, `LIMIT` o `OFFSET` al livello principale.
- La definizione della vista non deve contenere operazioni sugli insiemi (`UNION`, `INTERSECT` o`EXCEPT`) al livello principale.
- L'elenco di selezione della vista non deve contenere aggregati, funzioni finestra o funzioni che restituiscono insiemi.

Una vista aggiornabile automaticamente può contenere una combinazione di colonne aggiornabili e non aggiornabili. Una colonna è aggiornabile se è un semplice riferimento a una colonna aggiornabile della relazione di base sottostante. In caso contrario, la colonna è di sola lettura e si verifica un errore se un'istruzione `INSERT` o `UPDATE` tenta di assegnarle un valore.

Per le viste aggiornabili automaticamente, il sistema converte qualsiasi istruzione `INSERT`, `UPDATE` o `DELETE` sulla vista nell'istruzione corrispondente sulla relazione di base sottostante. Le istruzioni `INSERT` con una clausola `ON CONFLICT UPDATE` sono pienamente supportate.

Se una vista aggiornabile automaticamente contiene una condizione `WHERE`, la condizione limita le righe della relazione di base che possono essere modificate dalle istruzioni `UPDATE` e `DELETE` eseguite sulla vista. Tuttavia, un'istruzione `UPDATE` può modificare una riga in modo che non soddisfi più la condizione `WHERE`, rendendola invisibile dalla vista. Allo stesso modo, un comando `INSERT` può potenzialmente inserire righe nella relazione di base che non soddisfano la condizione `WHERE`, rendendole invisibili attraverso la vista. `ON CONFLICT UPDATE` può influire in modo analogo su una riga esistente non visibile attraverso la vista.

È possibile utilizzare `CHECK OPTION` per impedire che i comandi `INSERT` e `UPDATE` creino righe che non sono visibili attraverso la vista.

Se una vista aggiornabile automaticamente è contrassegnata con la proprietà security_barrier, tutte le condizioni `WHERE` della vista (e tutte le condizioni che utilizzano gli operatori contrassegnati come `LEAKPROOF`) vengono sempre valutate prima di qualsiasi condizione aggiunta da un utente della

vista. Bisogne tenere presente che, per questo motivo, le righe che alla fine non vengono restituite (perché non soddisfano le condizioni `WHERE` dell'utente) potrebbero comunque finire per essere bloccate. È possibile utilizzare `EXPLAIN` per vedere quali condizioni vengono applicate a livello di relazione (e quindi non bloccano le righe) e quali no.

Una vista più complessa che non soddisfa tutte queste condizioni è di sola lettura per impostazione predefinita: il sistema non consente inserimenti, aggiornamento o eliminazioni sulla vista.

> **Note**
>
> L'utente che esegue l'inserimento, l'aggiornamento o l'eliminazione sulla vista deve disporre del privilegio di inserimento, aggiornamento o eliminazione corrispondente sulla vista. Per impostazione predefinita, il proprietario della vista deve disporre dei privilegi pertinenti sulle relazioni di base sottostanti, mentre l'utente che esegue l'aggiornamento non necessita di alcuna autorizzazione sulle relazioni di base sottostanti. Tuttavia, se la vista ha l'opzione security_invoker impostata su true, è l'utente che esegue l'aggiornamento, anziché il proprietario della vista, che deve disporre dei privilegi pertinenti sulle relazioni di base sottostanti.

Esempi

Per creare una visualizzazione composta da tutti i film comici.

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Questa istruzione creerà una vista contenente le colonne presenti nella tabella `film` al momento della creazione della vista. Sebbene sia stato utilizzato il simbolo `*` per creare la vista, le colonne aggiunte successivamente alla tabella non faranno parte della vista.

Creazione di una vista con `LOCAL CHECK OPTION`.

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
    WITH CASCADED CHECK OPTION;
```

Questa istruzione creerà una vista che controlla `kind` e `classification` delle nuove righe.

Creazione di una vista con una combinazione di colonne aggiornabili e non aggiornabili.

```
CREATE VIEW comedies AS
    SELECT f.*,
           country_code_to_name(f.country_code) AS country,
           (SELECT avg(r.rating)
            FROM user_ratings r
            WHERE r.film_id = f.id) AS avg_rating
    FROM films f
    WHERE f.kind = 'Comedy';
```

Questa vista supporta `INSERT`, `UPDATE` e `DELETE`. Tutte le colonne della tabella dei film saranno aggiornabili, mentre le colonne calcolate `country` e `avg_rating` saranno di sola lettura.

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

> **Note**
>
> Sebbene il nome della vista ricorsiva sia qualificato dallo schema in questo comando `CREATE`, il suo autoreferenziamento interno non è qualificato dallo schema. Questo perché il nome Common Table Expression (CTE) creato implicitamente non può essere qualificato dallo schema.

Compatibilità

`CREATE OR REPLACE VIEW` è un'estensione del linguaggio PostgreSQL. Anche la `WITH ( ... )` clausola è un'estensione, così come le viste con guardabarriere e le viste con security invoker. Aurora DSQL supporta queste estensioni del linguaggio.

ALTER VIEW

L'istruzione `ALTER VIEW` consente di modificare varie proprietà di una vista esistente e Aurora DSQL supporta tutta la sintassi PostgreSQL per questo comando.

Sintassi supportata

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER VIEW [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER VIEW [ IF EXISTS ] name RENAME [ COLUMN ] column_name TO new_column_name
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

Description

`ALTER VIEW` modifica varie proprietà ausiliarie di una vista. (se si intende modificare la query di definizione della vista, utilizzare `CREATE OR REPLACE VIEW`). Per utilizzare `ALTER VIEW` è necessario essere proprietari della vista da utilizzare. Per modificare lo schema di una vista, occorre anche disporre del privilegio `CREATE` sul nuovo schema. Per modificare il proprietario, bisogna essere in grado di eseguire l'istruzione `SET ROLE` sul nuovo ruolo proprietario e tale ruolo deve disporre del privilegio `CREATE` sullo schema della vista. Queste restrizioni impongono che la modifica del proprietario non comporti alcun effetto che non si possa ottenere eliminando e ricreando la visualizzazione.

Parameters

Parametri di `ALTER VIEW`

`name`

Il nome (facoltativamente qualificato dallo schema) di una vista esistente.

`column_name`

Nuovo nome per una colonna esistente.

`IF EXISTS`

Non generare un errore se la vista non esiste. In questo caso viene emesso un avviso.

`SET/DROP DEFAULT`

Questi moduli impostano o rimuovono il valore predefinito per una colonna. Il valore predefinito per una colonna di visualizzazione viene sostituito in qualsiasi comando `INSERT` o `UPDATE` in cui

la destinazione è la vista. Il valore predefinito per la vista avrà la precedenza su qualsiasi valore predefinito delle relazioni sottostanti.

new_owner

Il nome utente del nuovo proprietario della vista.

new_name

Il nuovo nome della vista.

new_schema

Il nuovo schema della vista.

SET (view_option_name [= view_option_value] [, ...]), **RESET (view_option_name [, ...])**

Imposta o reimposta un'opzione della vista. Di seguito sono riportate le opzioni supportate.

- `check_option (enum)` - Modifica l'opzione di controllo della vista. Il valore deve essere `local` o `cascaded`.
- `security_barrier (boolean)` - Modifica la proprietà guardabarriere della vista. Il valore deve essere un valore booleano, come `true` o `false`.
- `security_invoker (boolean)` - Modifica la proprietà guardabarriere della vista. Il valore deve essere un valore booleano, come `true` o `false`.

Note

Per ragioni storiche di PostgreSQL, `ALTER TABLE` può essere utilizzato anche sulle viste, ma le uniche varianti di `ALTER TABLE` consentite sulle viste sono equivalenti a quelle mostrate in precedenza.

Esempi

Ridenominazione della vista `foo` in `bar`.

```
ALTER VIEW foo RENAME TO bar;
```

Associazione di un valore di colonna predefinito a una vista aggiornabile.

```
CREATE TABLE base_table (id int, ts timestamptz);
```

```
CREATE VIEW a_view AS SELECT * FROM base_table;
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();
INSERT INTO base_table(id) VALUES(1);  -- ts will receive a NULL
INSERT INTO a_view(id) VALUES(2);  -- ts will receive the current time
```

Compatibilità

`ALTER VIEW` è un'estensione PostgreSQL dello standard SQL supportato da Aurora DSQL.

DROP VIEW

L'istruzione `DROP VIEW` rimuove una vista esistente. Aurora DSQL supporta la sintassi PostgreSQL completa per questo comando.

Sintassi supportata

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Description

`DROP VIEW` elimina una vista esistente. Per eseguire questo comando è necessario essere proprietari della vista.

Parameters

IF EXISTS

Non generare un errore se la vista non esiste. In questo caso viene emesso un avviso.

name

Il nome (facoltativamente qualificato dallo schema) della vista da rimuovere.

CASCADE

Rilascia automaticamente gli oggetti che dipendono dalla vista (come le altre viste) e, a loro volta, tutti gli oggetti che dipendono da tali oggetti.

RESTRICT

Rifiuta di eliminare la vista se alcuni oggetti dipendono da essa. Questa è l'impostazione predefinita.

Esempi

```
DROP VIEW kinds;
```

Compatibilità

Questo comando è conforme allo standard SQL, tranne per il fatto che lo standard consente di eliminare una sola vista per comando e a parte l'opzione `IF EXISTS`, che è un'estensione PostgreSQL supportata da Aurora DSQL.

Migrazione da PostgreSQL ad Aurora SQL

Aurora DSQL è progettata per essere compatibile con PostgreSQL e supporta funzionalità relazionali di base come transazioni ACID, indici secondari, join e operazioni DML standard. La maggior parte delle applicazioni PostgreSQL esistenti può migrare ad Aurora DSQL con modifiche minime.

Questa sezione fornisce linee guida pratiche per la migrazione dell'applicazione ad Aurora DSQL, tra cui compatibilità del framework, modelli di migrazione e considerazioni sull'architettura.

Compatibilità con framework e ORM

Aurora DSQL utilizza il protocollo wire PostgreSQL standard, garantendo la compatibilità con i driver e i framework PostgreSQL. I più diffusi ORMs funzionano con Aurora DSQL con modifiche minime o nulle. Vedi the section called "Adattatori di Aurora DSQL" per le implementazioni di riferimento e le integrazioni ORM disponibili.

Modelli di migrazione comuni

Durante la migrazione da PostgreSQL ad Aurora DSQL, alcune funzionalità funzionano in modo diverso o hanno una sintassi alternativa. Questa sezione fornisce indicazioni sugli scenari di migrazione più comuni.

Alternative operative DDL

Aurora DSQL offre alternative moderne alle tradizionali operazioni DDL PostgreSQL:

Creazione di indici

- Utilizzatelo `CREATE INDEX ASYNC` al posto di quello `CREATE INDEX` per la creazione di indici non bloccanti.

  Vantaggio: creazione di indici senza tempi di inattività su tabelle di grandi dimensioni.

Rimozione dei dati

Utilizzare `DELETE FROM table_name` al posto di`TRUNCATE`.

Alternativa: per una completa ricreazione a tavola, utilizzare `DROP TABLE` seguito da`CREATE TABLE`.

Configurazione del sistema

Aurora DSQL non supporta `ALTER SYSTEM` i comandi perché il sistema è completamente gestito. La configurazione viene gestita automaticamente in base ai modelli di carico di lavoro.

Vantaggio: non è necessario ottimizzare il database o gestire i parametri.

Modelli di progettazione dello schema

Adatta questi modelli PostgreSQL comuni per la compatibilità con Aurora SQL:

Sequenze per chiavi

Usa le UUIDs nostre chiavi composite invece di sequenze ad incremento automatico. Le sequenze ad incremento automatico portano a un'elevata quantità di conflitti in un sistema distribuito poiché più autori cercano di aggiornare gli stessi dati. UUIDs forniscono la stessa funzione ma non richiedono alcun coordinamento.

Esempio: `id UUID PRIMARY KEY DEFAULT gen_random_uuid()`

modelli di integrità referenziale

Aurora DSQL supporta le relazioni e le `JOIN` operazioni tra tabelle ma non impone ancora vincoli di chiave esterna. Questa scelta progettuale si allinea ai moderni modelli di database distribuiti in cui la convalida a livello di applicazione offre maggiore flessibilità ed evita i colli di bottiglia delle prestazioni dovuti alle operazioni a cascata.

Modello: implementa i controlli di integrità referenziale a livello di applicazione utilizzando convenzioni di denominazione, logica di convalida e limiti di transazione coerenti. Molte applicazioni su larga scala preferiscono questo approccio per un migliore controllo sulla gestione degli errori e sulle prestazioni.

Gestione temporanea dei dati

Utilizza CTEs sottoquery o tabelle normali con logica di pulizia anziché tabelle temporanee.

Alternativa: crea tabelle con nomi specifici della sessione e puliscile nell'applicazione.

Comprendere le differenze architettoniche

L'architettura distribuita e serverless di Aurora DSQL si differenzia intenzionalmente da PostgreSQL tradizionale in diverse aree. Queste differenze consentono i principali vantaggi di semplicità e scalabilità di Aurora DSQL.

Modello di database semplificato

Database singolo per cluster

Aurora DSQL fornisce un database integrato denominato `postgres` per cluster.

Suggerimento per la migrazione: se l'applicazione utilizza più database, create cluster Aurora DSQL separati per la separazione logica o utilizzate schemi all'interno di un singolo cluster.

Nessuna tabella temporanea

Le tabelle temporanee non sono ancora supportate in Aurora DSQL. Le espressioni di tabella comuni (CTEs) e le sottoquery possono essere utilizzate come alternativa alle query complesse.

Alternativa: da utilizzare CTEs con `WITH` clausole per set di risultati temporanei o tabelle normali con denominazione univoca per dati specifici della sessione.

Gestione automatica dello storage

Aurora DSQL elimina i tablespace e la gestione manuale dello storage. Lo storage si ridimensiona e si ottimizza automaticamente in base ai modelli di dati.

Vantaggio: non è necessario monitorare lo spazio su disco, pianificare l'allocazione dello storage o gestire le configurazioni dei tablespace.

Modelli di applicazione moderni

Aurora DSQL incoraggia modelli di sviluppo di applicazioni moderni che migliorano la manutenibilità e le prestazioni:

Logica a livello di applicazione anziché trigger di database

Aurora DSQL non supporta i trigger.

Strategia di migrazione: sposta la logica dei trigger nel codice dell'applicazione, usa architetture basate sugli eventi con AWS servizi come EventBridge o implementa gli audit trail utilizzando la registrazione delle applicazioni.

Funzioni SQL per l'elaborazione dei dati

Aurora DSQL supporta funzioni basate su SQL ma non linguaggi procedurali come PL/pgSQL.

Alternativa: utilizza le funzioni SQL per le trasformazioni dei dati o sposta la logica complessa sul livello dell'applicazione o sulle funzioni AWS Lambda.

Controllo ottimistico della concorrenza anziché blocco pessimistico

Aurora DSQL utilizza il controllo ottimistico della concorrenza (OCC), un approccio privo di blocchi che si differenzia dai tradizionali meccanismi di blocco del database. Invece di acquisire blocchi che bloccano altre transazioni, Aurora DSQL consente alle transazioni di procedere senza blocchi e rileva i conflitti al momento del commit. Ciò elimina le situazioni di stallo e impedisce che le transazioni lente blocchino altre operazioni.

Differenza fondamentale: quando si verificano conflitti, Aurora DSQL restituisce un errore di serializzazione anziché far attendere le transazioni per i blocchi. Ciò richiede che le applicazioni implementino una logica di ripetizione, simile alla gestione dei timeout di blocco nei database tradizionali, ma i conflitti vengono risolti immediatamente anziché causare attese di blocco.

Modello di progettazione: implementa una logica di transazione idempotente con meccanismi di ripetizione. Progetta schemi per ridurre al minimo i conflitti utilizzando chiavi primarie casuali e distribuendo gli aggiornamenti su tutta la gamma di chiavi. Per informazioni dettagliate, vedi Controllo della concorrenza in Aurora DSQL.

Relazioni e integrità referenziale

Aurora DSQL supporta le relazioni con chiavi esterne tra le tabelle, comprese le `JOIN` operazioni, ma i vincoli di chiave esterna non sono ancora supportati. Sebbene l'applicazione dell'integrità referenziale possa essere utile, le operazioni a cascata (come le eliminazioni a cascata) possono creare problemi di prestazioni imprevisti, ad esempio l'eliminazione di un ordine con 1.000 voci diventa una transazione di 1.001 righe. Per questo motivo, molti clienti evitano i vincoli relativi alle chiavi esterne.

Modello di progettazione: implementa i controlli di integrità referenziale a livello applicativo, utilizza eventuali modelli di coerenza o sfrutta AWS i servizi per la convalida dei dati.

Semplificazioni operative

Aurora DSQL elimina molte attività tradizionali di manutenzione del database, riducendo il sovraccarico operativo:

Non è richiesta alcuna manutenzione manuale

Aurora SQL non richiede `VACUUM` comandi`TRUNCATE`. `ALTER SYSTEM` Il sistema gestisce automaticamente l'ottimizzazione dello storage, la raccolta di statistiche e l'ottimizzazione delle prestazioni.

Vantaggio: elimina la necessità di finestre di manutenzione del database, pianificazione a vuoto e ottimizzazione dei parametri di sistema.

Partizionamento e scalabilità automatici

Aurora DSQL partiziona e distribuisce automaticamente i dati in base ai modelli di accesso. Il partizionamento e le sequenze manuali non sono necessari.

Suggerimento per la migrazione: rimuovete la logica di partizionamento manuale e lasciate che Aurora DSQL gestisca la distribuzione dei dati. Usa UUIDs o generati dall'applicazione anziché sequenze IDs .

Migrazione assistita dall'intelligenza artificiale

Puoi sfruttare gli strumenti di intelligenza artificiale per aiutarti a migrare la tua codebase su Aurora DSQL:

Utilizzo di Kiro per l'assistenza alla migrazione

Gli agenti di codifica come Kiro possono aiutarti ad analizzare e migrare il codice PostgreSQL su Aurora DSQL:

- Analisi dello schema: carica i file di schema esistenti e chiedi a Kiro di identificare potenziali problemi di compatibilità e suggerire alternative
- Trasformazione del codice: fornisci il codice dell'applicazione e chiedi a Kiro di aiutarti a rifattorizzare la logica di attivazione, sostituire le sequenze con o modificare i modelli di UUIDs transazione
- Pianificazione della migrazione: chiedi a Kiro di creare un piano di step-by-step migrazione basato sulla tua architettura applicativa specifica

Esempi di istruzioni Kiro:

```
"Analyze this PostgreSQL schema for DSQL compatibility and suggest alternatives for any
 unsupported features"
```

```
"Help me refactor this trigger function into application-level logic for DSQL
 migration"

"Create a migration checklist for moving my Django application from PostgreSQL to DSQL"
```

Server MCP Aurora DSQL

Il server Aurora DSQL Model Context Protocol (MCP) consente agli assistenti AI come Claude di connettersi direttamente al cluster Aurora DSQL e di cercare la documentazione di Aurora DSQL. Ciò consente all'IA di:

- Analizza lo schema esistente e suggerisci modifiche alla migrazione
- Esegui il test delle query e verifica la compatibilità durante la migrazione
- Fornisci up-to-date linee guida accurate basate sulla più recente documentazione di Aurora DSQL

Per utilizzare il server MCP Aurora DSQL con Claude o altri assistenti AI, consulta le istruzioni di configurazione per il server MCP Aurora DSQL.

Considerazioni su Aurora DSQL rispetto alla compatibilità con PostgreSQL

Aurora DSQL presenta differenze nel supporto delle funzionalità rispetto a PostgreSQL autogestito che ne consentono l'architettura distribuita, il funzionamento senza server e la scalabilità automatica. La maggior parte delle applicazioni funziona entro queste differenze senza modifiche.

Per le considerazioni generali, consulta Considerazioni sull'utilizzo di Amazon Aurora DSQL. Per quote e limiti, consulta Quote di cluster e limiti del database in Amazon Aurora DSQL.

- Aurora DSQL utilizza un unico database integrato denominato `postgres`. Non è possibile creare database aggiuntivi o rinominare o eliminare il database `postgres`.
- Il database `postgres` utilizza la codifica caratteri UTF-8. Non è possibile modificare la codifica del server.
- Il database utilizza solo le regole di confronto `C`.
- Aurora DSQL utilizza `UTC` come fuso orario del sistema. Postgres memorizza internamente tutte le date e gli orari in base al fuso orario in UTC. È possibile impostare il parametro di `TimeZone` configurazione per convertire il modo in cui viene visualizzato al client e fungere da impostazione predefinita per l'input del client che il server utilizzerà per la conversione in UTC internamente.
- Il livello di isolamento delle transazioni è fisso su PostgreSQL `Repeatable Read`.

- Le transazioni sono soggette ai seguenti vincoli:
  - Una transazione non può combinare operazioni DDL e DML
  - Una transazione può includere solo 1 istruzione DDL
  - Una transazione può modificare fino a 3.000 righe, indipendentemente dal numero di indici secondari
  - Il limite di 3.000 righe si applica a tutte le istruzioni DML (`INSERT`, `UPDATE`, `DELETE`)
- Le connessioni al database scadono dopo 1 ora.
- Aurora DSQL attualmente non consente l'esecuzione di `GRANT [permission] ON DATABASE`. Se si tenta di eseguire tale istruzione, Aurora DSQL restituisce il messaggio di errore `ERROR: unsupported object type in GRANT`.
- Aurora DSQL non consente ai ruoli utente non amministratori di eseguire il comando `CREATE SCHEMA`. Non è possibile eseguire il comando `GRANT [permission] on DATABASE` e concedere le autorizzazioni `CREATE` sul database. Se un ruolo utente non amministratore tenta di creare uno schema, Aurora DSQL restituisce il messaggio di errore `ERROR: permission denied for database postgres`.
- Gli utenti non amministratori non possono creare oggetti nello schema pubblico. Solo gli utenti amministratori possono creare oggetti nello schema pubblico. Il ruolo utente amministratore dispone delle autorizzazioni per concedere l'accesso in lettura, scrittura e modifica a questi oggetti a utenti non amministratori, ma non può concedere le autorizzazioni `CREATE` sullo schema pubblico stesso. Gli utenti non amministratori devono utilizzare schemi diversi creati dall'utente per la creazione di oggetti.

Hai bisogno di aiuto con la migrazione?

Se riscontri funzionalità fondamentali per la migrazione ma attualmente non supportate in Aurora DSQL, consulta Fornire feedback su Amazon Aurora DSQL per informazioni su come condividere il feedback con AWS.

Controllo della concorrenza in Aurora DSQL

La concorrenza consente a più sessioni di accedere e modificare i dati contemporaneamente senza compromettere l'integrità e la coerenza dei dati. Aurora DSQL offre la compatibilità con PostgreSQL implementando al contempo un meccanismo di controllo della concorrenza moderno e senza blocchi. Mantiene la piena conformità ACID attraverso l'isolamento degli snapshot, garantendo la coerenza e l'affidabilità dei dati.

Un vantaggio chiave di Aurora DSQL è la sua architettura priva di blocchi, che elimina i comuni colli di bottiglia nelle prestazioni del database. Aurora DSQL impedisce che le transazioni lente blocchino altre operazioni ed elimina il rischio di deadlock. Questo approccio rende Aurora DSQL particolarmente utile per applicazioni ad alto throughput, in cui le prestazioni e la scalabilità sono fondamentali.

Conflitti tra transazioni

Aurora DSQL utilizza il controllo ottimistico della concorrenza (OCC), che funziona in modo diverso dai tradizionali sistemi basati su blocchi. Invece di utilizzare i blocchi, OCC valuta i conflitti al momento del commit. Quando più transazioni entrano in conflitto durante l'aggiornamento della stessa riga, Aurora SQL gestisce le transazioni come segue:

- La transazione con il tempo di commit più breve viene elaborata da Aurora DSQL.
- Le transazioni in conflitto ricevono un errore di serializzazione PostgreSQL, che indica la necessità di riprovare.

È opportuno progettare le applicazioni in modo da implementare una logica di ripetizione per gestire i conflitti. Il modello di progettazione ideale è idempotente e consente di ripetere la transazione come primo rimedio, quando possibile. La logica consigliata è simile alla logica abort and retry in una situazione di timeout o deadlock di PostgreSQL standard. Tuttavia, OCC richiede che le applicazioni applichino questa logica più frequentemente.

Linee guida per l'ottimizzazione delle prestazioni delle transazioni

Per ottimizzare le prestazioni, è opportuno ridurre al minimo contese elevate su chiavi singole o intervalli di chiavi ridotti. Per raggiungere questo obiettivo, progetta lo schema in modo da distribuire gli aggiornamenti sull'intervallo di chiavi del cluster utilizzando le seguenti linee guida:

- Scegli una chiave primaria casuale per le tabelle.
- Evita i modelli che fanno aumentare la contesa sulle singole chiavi. Questo approccio garantisce prestazioni ottimali anche con l'aumento del volume delle transazioni.

DDL e transazioni distribuite in Aurora DSQL

Il DDL (Data Definition Language) si comporta in modo diverso in Aurora DSQL rispetto a PostgreSQL. Aurora DSQL offre un livello di database Multi-AZ distribuito e senza condivisione

basato su parchi di risorse di calcolo e archiviazione multi-tenant. Poiché non esiste un singolo nodo o leader del database primario, il catalogo del database viene distribuito. Pertanto, Aurora DSQL gestisce le modifiche DDL allo schema come transazioni distribuite.

In particolare, DDL si comporta in modo diverso in Aurora DSQL come segue:

Errori di controllo della concorrenza

Aurora DSQL restituisce un errore di violazione del controllo della concorrenza se si esegue una transazione mentre un'altra transazione aggiorna una risorsa. Considera, ad esempio, la seguente sequenza di azioni:

1. Nella sessione 1, un utente aggiunge una colonna alla tabella `mytable`.
2. Nella sessione 2, un utente tenta di inserire una riga in `mytable`.

   Aurora DSQL restituisce l'errore `SQL Error [40001]: ERROR: schema has been updated by another transaction, please retry: (OC001).`

DDL e DML nella stessa transazione

Le transazioni in Aurora DSQL possono contenere solo un'istruzione DDL e non possono avere sia istruzioni DDL che DML. Questa restrizione significa che non è possibile creare una tabella e inserire dati nella stessa tabella all'interno della stessa transazione. Ad esempio, Aurora DSQL supporta le seguenti transazioni sequenziali.

```
BEGIN;
  CREATE TABLE mytable (ID_col integer);
COMMIT;

BEGIN;
  INSERT into FOO VALUES (1);
COMMIT;
```

Aurora DSQL non supporta la seguente transazione, che include contemporaneamente istruzioni `CREATE` e `INSERT`.

```
BEGIN;
  CREATE TABLE FOO (ID_col integer);
  INSERT into FOO VALUES (1);
COMMIT;
```

DDL asincrono

In PostgreSQL standard, le operazioni DDL come `CREATE INDEX` bloccano la tabella interessata, rendendola non disponibile per le letture e le scritture da altre sessioni. In Aurora DSQL, queste istruzioni DDL vengono eseguite in modo asincrono utilizzando un gestore in background. L'accesso alla tabella interessata non è bloccato. Pertanto, istruzioni DDL su tabelle di grandi dimensioni possono esseri eseguito senza tempo di inattività o impatto sulle prestazioni. Per maggiori informazioni sull'utilizzo dello strumento di gestione dei processi asincroni in Aurora DSQL, consulta Indici asincroni in Aurora SQL.

Chiavi primarie in Aurora DSQL

In Aurora DSQL, una chiave primaria è una funzionalità che organizza fisicamente i dati delle tabelle. È simile all'operazione `CLUSTER` in PostgreSQL o a un indice cluster in altri database. Quando si definisce una chiave primaria, Aurora DSQL crea un indice che include tutte le colonne della tabella. La struttura a chiave primaria di Aurora DSQL garantisce un accesso e una gestione efficienti dei dati.

Struttura e archiviazione dei dati

Quando si definisce una chiave primaria, Aurora DSQL memorizza i dati della tabella nell'ordine delle chiavi primarie. Questa struttura organizzata a indice consente una ricerca tramite chiave primaria per recuperare direttamente tutti i valori delle colonne, invece di seguire un puntatore ai dati come in un tradizionale indice B-tree. A differenza dell'operazione `CLUSTER` in PostgreSQL, che riorganizza i dati una sola volta, Aurora DSQL mantiene questo ordine automaticamente e continuamente. Questo approccio migliora le prestazioni delle query che si basano sull'accesso alla chiave primaria.

Aurora DSQL utilizza anche la chiave primaria per generare una chiave unica a livello di cluster per ogni riga di tabelle e indici. Questa chiave unica è anche alla base della gestione distribuita dei dati. Consente il partizionamento automatico dei dati su più nodi, supportando uno storage scalabile e un'elevata concorrenza. Di conseguenza, la struttura a chiave primaria permette ad Aurora DSQL di scalare automaticamente e di gestire i carichi di lavoro simultanei in modo efficiente.

Linee guida per la scelta di una chiave primaria

Quando si sceglie e si utilizza una chiave primaria in Aurora DSQL, bisogna tenere presenti le seguenti linee guida:

- Definire una chiave primaria quando si crea una tabella. Non è possibile modificare questa chiave né aggiungere una nuova chiave primaria in un secondo momento. La chiave primaria diventa parte della chiave a livello di cluster utilizzata per il partizionamento dei dati e il dimensionamento automatico del throughput di scrittura. Se non si specifica una chiave primaria, Aurora DSQL assegna un ID sintetico nascosto.
- Per le tabelle con volumi di scrittura elevati, evitare di utilizzare numeri interi che aumentano in modo monotono come chiavi primarie. Ciò può causare problemi di prestazioni se si indirizzano tutti i nuovi inserimenti su un'unica partizione. Utilizzare invece chiavi primarie con distribuzione casuale per garantire una distribuzione uniforme delle scritture tra le partizioni di archiviazione.
- Per le tabelle che vengono modificate raramente o sono di sola lettura, è possibile utilizzare una chiave crescente. Esempi di chiavi crescenti sono i timestamp o i numeri di sequenza. Una chiave densa contiene molti valori ravvicinati o duplicati. È possibile utilizzare una chiave crescente anche se è densa, poiché le prestazioni di scrittura sono meno importanti.
- Se una scansione completa della tabella non soddisfa i requisiti di prestazione, scegliere un metodo di accesso più efficiente. Nella maggior parte dei casi, ciò significa utilizzare una chiave primaria che corrisponda alla chiave di join e lookup più comune nelle query.
- La dimensione massima combinata delle colonne in una chiave primaria è 1 kibibyte. Per maggiori informazioni, consulta Limiti del database in Aurora DSQL e Tipi di dati supportati in Aurora DSQL.
- È possibile includere fino a 8 colonne in una chiave primaria o in un indice secondario. Per maggiori informazioni, consulta Limiti del database in Aurora DSQL e Tipi di dati supportati in Aurora DSQL.

Indici asincroni in Aurora SQL

Il comando `CREATE INDEX ASYNC` crea un indice su una o più colonne di una tabella specificata. Questo comando è un'operazione DDL asincrona che non blocca altre transazioni. Quando si esegue il comando `CREATE INDEX ASYNC`, Aurora DSQL restituisce immediatamente un `job_id`.

È possibile monitorare lo stato di questo processo asincrono utilizzando la vista di sistema `sys.jobs`. Mentre il processo di creazione dell'indice è in corso, è possibile utilizzare le procedure e i comandi seguenti:

`sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`*

Bloccare la sessione corrente fino al completamento o all'esito negativo del processo specificato. Restituisce un valore booleano che indica la riuscita o l'errore.

`DROP INDEX`

Annullare un processo di creazione dell'indice in esecuzione.

Al termine della creazione dell'indice asincrono, Aurora DSQL aggiorna il catalogo di sistema per contrassegnare l'indice come attivo.

> **Note**
>
> Tieni presente che le transazioni simultanee che accedono a oggetti nello stesso namespace durante questo aggiornamento potrebbero riscontrare errori di concorrenza.

Quando Aurora DSQL termina un'attività di indicizzazione asincrona, aggiorna il catalogo di sistema per mostrare che l'indice è attivo. Se altre transazioni fanno riferimento agli oggetti nello stesso namespace in tale momento, potrebbero visualizzare un errore di concorrenza.

Sintassi

`CREATE INDEX ASYNC` utilizza la seguente sintassi.

```
CREATE [ UNIQUE ] INDEX ASYNC [ IF NOT EXISTS ] name ON table_name
     ( { column_name } [ NULLS { FIRST | LAST } ] )
     [ INCLUDE ( column_name [, ...] ) ]
     [ NULLS [ NOT ] DISTINCT ]
```

Parameters

`UNIQUE`

Indica ad Aurora DSQL di verificare la presenza di valori duplicati nella tabella quando crea l'indice e ogni volta che si aggiungono dati. Se si specifica questo parametro, le operazioni di inserimento e aggiornamento che comporterebbero la duplicazione delle voci generano un errore.

`IF NOT EXISTS`

Indica che Aurora DSQL non deve generare un'eccezione se esiste già un indice con lo stesso nome. In questa situazione, Aurora DSQL non crea il nuovo indice. Nota che l'indice che stai cercando di creare potrebbe avere una struttura molto diversa dall'indice esistente. Se non viene specificato questo parametro, il nome dell'indice è obbligatorio.

name

Il nome dell'indice. Non è possibile includere il nome dello schema in questo parametro.

Aurora DSQL crea l'indice nello stesso schema della tabella principale. Il nome dell'indice deve essere distinto dal nome di qualsiasi altro oggetto, ad esempio una tabella o un indice, nello schema.

Se non si specifica un nome, Aurora DSQL genera automaticamente un nome basato sul nome della tabella principale e della colonna indicizzata. Ad esempio, se si esegue `CREATE INDEX ASYNC on table1 (col1, col2)`, Aurora DSQL assegna automaticamente un nome all'indice `table1_col1_col2_idx`.

`NULLS FIRST | LAST`

Criterio di ordinamento delle colonne nulle e non nulle. `FIRST` indica che Aurora DSQL deve ordinare le colonne nulle prima delle colonne non nulle. `LAST` indica che Aurora DSQL deve ordinare le colonne nulle dopo le colonne non nulle.

`INCLUDE`

Un elenco di colonne da includere nell'indice come colonne non chiave. Non è possibile utilizzare una colonna non chiave in una qualifica di ricerca basata sulla scansione dell'indice. Aurora DSQL ignora la colonna in termini di unicità di un indice.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Specifica se Aurora DSQL deve considerare i valori null come distinti in un indice univoco. L'impostazione predefinita è `DISTINCT`, il che significa che un indice univoco può contenere più valori nulli in una colonna. `NOT DISTINCT` indica che un indice non può contenere più valori nulli in una colonna.

Note per l'utilizzo

Considera le linee guida seguenti:

- Il comando `CREATE INDEX ASYNC` non introduce blocchi. Inoltre, non influisce sulla tabella di base utilizzata da Aurora DSQL per creare l'indice.
- Durante le operazioni di migrazione dello schema, è utile la procedura `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`*. Garantisce che le operazioni DDL e DML successive abbiano come target l'indice appena creato.

- Ogni volta che Aurora DSQL esegue una nuova attività asincrona, controlla la vista `sys.jobs` ed elimina le attività con uno stato di `completed` o `failed` pari o superiore a 30 minuti. Pertanto, `sys.jobs` mostra principalmente le attività in corso e non contiene informazioni sulle attività precedenti.
- Se Aurora DSQL non riesce a creare un indice asincrono, l'indice rimane in stato `INVALID`. Per gli indici univoci, le operazioni DML sono soggette a vincoli di unicità finché non si elimina l'indice. Si consiglia di eliminare gli indici non validi e di ricrearli.

Creazione di un indice: esempio

L'esempio seguente mostra come creare uno schema, una tabella e quindi un indice.

1. Crea una nuova tabella denominata `test.departments`.

```
CREATE SCHEMA test;

CREATE TABLE test.departments (name varchar(255) primary key NOT null,
     manager varchar(255),
     size varchar(4));
```

2. Inserisci una riga di dati nella tabella.

```
INSERT INTO test.departments VALUES ('Human Resources', 'John Doe', '10')
```

3. Crea un indice asincrono.

```
CREATE INDEX ASYNC test_index on test.departments(name, manager, size);
```

   Il comando `CREATE INDEX` restituisce un ID di processo, come mostrato di seguito.

```
job_id
-------------------------
jh2gbtx4mzhgfkbimtgwn5j45y
```

   `job_id` indica che Aurora DSQL ha avviato un nuovo processo per creare l'indice. È possibile utilizzare la procedura `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* per bloccare altri lavori sulla sessione fino al termine o al timeout del processo.

Esecuzione di query sullo stato di creazione dell'indice: esempio

Eseguire la query sulla vista di sistema `sys.jobs` per verificare lo stato di creazione dell'indice, come illustrato nell'esempio seguente.

```
SELECT * FROM sys.jobs
```

Aurora DSQL restituisce una risposta simile alla seguente.

```
           job_id           |   status   | details
----------------------------+------------+---------
 vs3kcl3rt5ddpk3a6xcq57cmcy | completed  |
 ihbyw2aoirfnrdfoc4ojnlamoq | processing |
```

La colonna stato può assumere uno dei seguenti valori.

submitted	**processing**	**failed**	**completed**
L'attività è stata inviata, ma Aurora DSQL non ha ancora iniziato a elaborarla.	Aurora DSQL sta elaborando l'operazione.	L'attività non è andata a buon fine. Per maggiori informazioni, consulta la colonna dettagli. Se Aurora DSQL non è riuscita a creare l'indice, non rimuove automaticamente la sua definizione. È necessario rimuovere manualmente l'indice con il comando `DROP INDEX`.	Aurora DSQL

È inoltre possibile eseguire la query sullo stato dell'indice tramite le tabelle `pg_index` e `pg_class` del catalogo. In particolare, gli attributi `indisvalid` e `indisimmediate` possono indicare in che stato si trova l'indice. Mentre Aurora DSQL crea l'indice, lo stato iniziale è `INVALID`. Il flag

`indisvalid` dell'indice restituisce `FALSE` o `f`, che indica che l'indice non è valido. Se il flag restituisce `TRUE` o `t`, l'indice è pronto.

```
SELECT relname AS index_name, indisvalid as is_valid, pg_get_indexdef(indexrelid) AS
 index_definition
from pg_index, pg_class
WHERE pg_class.oid = indexrelid AND indrelid = 'test.departments'::regclass;
```

```
    index_name    | is_valid |
 index_definition
------------------+----------
+-------------------------------------------------------------------------------------------------
 department_pkey  |     t    | CREATE UNIQUE INDEX department_pkey ON test.departments
 USING btree_index (title) INCLUDE (name, manager, size)
 test_index1      |     t    | CREATE INDEX test_index1 ON test.departments USING
 btree_index (name, manager, size)
```

Errori di creazione dell'indice univoco

Se il processo di creazione dell'indice univoco asincrono mostra uno stato non riuscito con il dettaglio `Found duplicate key while validating index for UCVs`, significa che non è stato possibile creare un indice univoco a causa di violazioni del vincolo di unicità.

Come risolvere gli errori di creazione dell'indice univoco

1. Rimuovi tutte le righe della tabella principale che contengono voci duplicate per le chiavi specificate nell'indice secondario univoco.
2. Elimina l'indice non creato.
3. Esegui un nuovo comando create index.

Rilevamento delle violazioni di unicità nelle tabelle primarie

La seguente query SQL consente di identificare i valori duplicati in una colonna specificata della tabella. Ciò è particolarmente utile quando è necessario applicare l'univocità a una colonna che attualmente non è impostata come chiave primaria o non ha un vincolo univoco, come gli indirizzi e-mail in una tabella utente.

Gli esempi seguenti mostrano come creare una tabella di utenti di esempio, popolarla con dati di test contenenti duplicati noti e quindi eseguire la query di rilevamento.

Definire lo schema della tabella

```
-- Drop the table if it exists
DROP TABLE IF EXISTS users;

-- Create the users table with a simple integer primary key
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY,
    email VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Inserire dati di esempio che includono set di indirizzi e-mail duplicati

```
-- Insert sample data with explicit IDs
INSERT INTO users (user_id, email, first_name, last_name) VALUES
    (1, 'john.doe@example.com', 'John', 'Doe'),
    (2, 'jane.smith@example.com', 'Jane', 'Smith'),
    (3, 'john.doe@example.com', 'Johnny', 'Doe'),
    (4, 'alice.wong@example.com', 'Alice', 'Wong'),
    (5, 'bob.jones@example.com', 'Bob', 'Jones'),
    (6, 'alice.wong@example.com', 'Alicia', 'Wong'),
    (7, 'bob.jones@example.com', 'Robert', 'Jones');
```

Eseguire una query di rilevamento dei duplicati

```
-- Query to find duplicates
WITH duplicates AS (
    SELECT email, COUNT(*) as duplicate_count
    FROM users
    GROUP BY email
    HAVING COUNT(*) > 1
)
SELECT u.*, d.duplicate_count
FROM users u
INNER JOIN duplicates d ON u.email = d.email
ORDER BY u.email, u.user_id;
```

Visualizzare tutti i record con indirizzi e-mail duplicati

```
 user_id |         email          | first_name | last_name |         created_at
 | duplicate_count
---------+------------------------+------------+-----------
+----------------------------+-----------------
       4 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |               2
       6 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |               2
       1 | john.doe@example.com   | John       | Doe       | 2025-05-21 20:55:53.714432
 |               2
       3 | john.doe@example.com   | Johnny     | Doe       | 2025-05-21 20:55:53.714432
 |               2
(4 rows)
```

Se provassimo ora l'istruzione di creazione dell'indice, questa fallirebbe:

```
postgres=> CREATE UNIQUE INDEX ASYNC idx_users_email ON users(email);
      job_id
----------------------------
 ve32upmjz5dgdknpbleeca5tri
(1 row)

postgres=> select * from sys.jobs;
           job_id           |  status   |                       details
       |  job_type   | class_id | object_id |      object_name       |       start_time
       |      update_time
----------------------------+-----------
+-------------------------------------------------------+-------------+----------
+-----------+------------------------+------------------------+------------------------
 qpn6aqlkijgmzilyidcpwrpova | completed |
          | DROP        |     1259 |     26384 |                        | 2025-05-20
 00:47:10+00 | 2025-05-20 00:47:32+00
 ve32upmjz5dgdknpbleeca5tri | failed    | Found duplicate key while validating index
 for UCVs | INDEX_BUILD |     1259 |     26396 | public.idx_users_email | 2025-05-20
 00:49:49+00 | 2025-05-20 00:49:56+00
(2 rows)
```

Tabelle e comandi di sistema in Aurora DSQL

Consulta le sezioni seguenti per informazioni sulle tabelle e i cataloghi di sistema supportati in Aurora DSQL.

Tabelle di sistema

Aurora DSQL è compatibile con PostgreSQL, quindi molte tabelle e viste del catalogo di sistema di PostgreSQL esistono anche in Aurora DSQL.

Tabelle e viste importanti del catalogo di PostgreSQL

La tabella seguente descrive le tabelle e le viste più comuni che è possibile utilizzare in Aurora DSQL.

Nome	Description
`pg_namespace`	Informazioni su tutti gli schemi
`pg_tables`	Informazioni su tutte le tabelle
`pg_attribute`	Informazioni su tutti gli attributi
`pg_views`	Informazioni sulle viste (pre)definite
`pg_class`	Descrive tutte le tabelle, le colonne, gli indici e gli oggetti simili
`pg_stats`	Una vista sulle statistiche del planner
`pg_user`	Informazioni sugli utenti
`pg_roles`	Informazioni su utenti e gruppi
`pg_indexes`	Elenca tutti gli indici
`pg_constraint`	Elenca i vincoli sulle tabelle

Tabelle di catalogo supportate e non supportate

La tabella seguente indica quali tabelle sono supportate e non supportate in Aurora DSQL.

Name	Applicabile ad Aurora DSQL
`pg_aggregate`	No

Name	Applicabile ad Aurora DSQL
pg_am	Sì
pg_amop	No
pg_amproc	No
pg_attrdef	Sì
pg_attribute	Sì
pg_authid	No (utilizzare pg_roles)
pg_auth_members	Sì
pg_cast	Sì
pg_class	Sì
pg_collation	Sì
pg_constraint	Sì
pg_conversion	No
pg_database	No
pg_db_role_setting	Sì
pg_default_acl	Sì
pg_depend	Sì
pg_description	Sì
pg_enum	No
pg_event_trigger	No
pg_extension	No

Name	Applicabile ad Aurora DSQL
pg_foreign_data_wrapper	No
pg_foreign_server	No
pg_foreign_table	No
pg_index	Sì
pg_inherits	Sì
pg_init_privs	No
pg_language	No
pg_largeobject	No
pg_largeobject_metadata	Sì
pg_namespace	Sì
pg_opclass	No
pg_operator	Sì
pg_opfamily	No
pg_parameter_acl	Sì
pg_partitioned_table	No
pg_policy	No
pg_proc	No
pg_publication	No
pg_publication_namespace	No
pg_publication_rel	No

Name	Applicabile ad Aurora DSQL
pg_range	Sì
pg_replication_origin	No
pg_rewrite	No
pg_seclabel	No
pg_sequence	No
pg_shdepend	Sì
pg_shdescription	Sì
pg_shseclabel	No
pg_statistic	Sì
pg_statistic_ext	No
pg_statistic_ext_data	No
pg_subscription	No
pg_subscription_rel	No
pg_tablespace	No
pg_transform	No
pg_trigger	No
pg_ts_config	Sì
pg_ts_config_map	Sì
pg_ts_dict	Sì
pg_ts_parser	Sì

Name	Applicabile ad Aurora DSQL
pg_ts_template	Sì
pg_type	Sì
pg_user_mapping	No

Viste di sistema supportate e non supportate

La tabella seguente indica quali viste sono supportate e non supportate in Aurora DSQL.

Name	Applicabile ad Aurora DSQL
pg_available_extensions	No
pg_available_extension_versions	No
pg_backend_memory_contexts	Sì
pg_config	No
pg_cursors	No
pg_file_settings	No
pg_group	Sì
pg_hba_file_rules	No
pg_ident_file_mappings	No
pg_indexes	Sì
pg_locks	No
pg_matviews	No
pg_policies	No

Name	Applicabile ad Aurora DSQL
pg_prepared_statements	No
pg_prepared_xacts	No
pg_publication_tables	No
pg_replication_origin_status	No
pg_replication_slots	No
pg_roles	Sì
pg_rules	No
pg_seclabels	No
pg_sequences	No
pg_settings	Sì
pg_shadow	Sì
pg_shmem_allocations	Sì
pg_stats	Sì
pg_stats_ext	No
pg_stats_ext_exprs	No
pg_tables	Sì
pg_timezone_abbrevs	Sì
pg_timezone_names	Sì
pg_user	Sì
pg_user_mappings	No

Name	Applicabile ad Aurora DSQL
pg_views	Sì
pg_stat_activity	No
pg_stat_replication	No
pg_stat_replication_slots	No
pg_stat_wal_receiver	No
pg_stat_recovery_prefetch	No
pg_stat_subscription	No
pg_stat_subscription_stats	No
pg_stat_ssl	Sì
pg_stat_gssapi	No
pg_stat_archiver	No
pg_stat_io	No
pg_stat_bgwriter	No
pg_stat_wal	No
pg_stat_database	No
pg_stat_database_conflicts	No
pg_stat_all_tables	No
pg_stat_all_indexes	No
pg_statio_all_tables	No
pg_statio_all_indexes	No

Name	Applicabile ad Aurora DSQL
pg_statio_all_sequences	No
pg_stat_slru	No
pg_statio_user_tables	No
pg_statio_user_sequences	No
pg_stat_user_functions	No
pg_stat_user_indexes	No
pg_stat_progress_analyze	No
pg_stat_progress_basebackup	No
pg_stat_progress_cluster	No
pg_stat_progress_create_index	No
pg_stat_progress_vacuum	No
pg_stat_sys_indexes	No
pg_stat_sys_tables	No
pg_stat_xact_all_tables	No
pg_stat_xact_sys_tables	No
pg_stat_xact_user_functions	No
pg_stat_xact_user_tables	No
pg_statio_sys_indexes	No
pg_statio_sys_sequences	No
pg_statio_sys_tables	No

Name	Applicabile ad Aurora DSQL
pg_statio_user_indexes	No

Le viste sys.jobs e sys.iam_pg_role_mappings

Aurora DSQL supporta le seguenti viste di sistema:

`sys.jobs`

`sys.jobs` fornisce informazioni sullo stato dei processi asincroni. Ad esempio, dopo aver creato un indice asincrono, Aurora DSQL restituisce un `job_uuid`. È possibile utilizzare tale `job_uuid` con `sys.jobs` per cercare lo stato del processo.

```
SELECT * FROM sys.jobs WHERE job_id = 'example_job_uuid';

        job_id        |   status   | details
----------------------+------------+---------
 example_job_uuid | processing |
(1 row)
```

`sys.iam_pg_role_mappings`

La vista `sys.iam_pg_role_mappings` fornisce informazioni sulle autorizzazioni concesse agli utenti IAM. Ad esempio, se `DQSLDBConnect` è un ruolo IAM che fornisce ad Aurora DSQL l'accesso ai non amministratori e a un utente denominato `testuser` vengono concessi il ruolo `DQSLDBConnect` e le autorizzazioni corrispondenti, è possibile interrogare la vista `sys.iam_pg_role_mappings` per vedere a quali utenti sono concesse quali autorizzazioni.

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Utili interrogazioni sui metadati di sistema

Utilizza queste query per ottenere statistiche e metadati delle tabelle senza eseguire operazioni costose come la scansione completa della tabella.

Ottieni il numero stimato di righe per una tabella

Per ottenere il conteggio approssimativo delle righe in una tabella senza eseguire una scansione completa della tabella, utilizzate la seguente query:

```
SELECT reltuples FROM pg_class WHERE relname = 'table_name';
```

Il comando restituisce un output simile al seguente:

```
  reltuples
--------------
 9.993836e+08
```

Questo approccio è più efficiente rispetto alle tabelle `SELECT COUNT(*)` di grandi dimensioni in Aurora DSQL.

Il comando **ANALYZE**.

Il comando `ANALYZE` raccoglie statistiche sul contenuto delle tabelle nel database e archivia i risultati nella vista di sistema `pg_stats`. In un secondo momento, il pianificatore di query utilizza queste statistiche per determinare i piani di esecuzione più efficienti per le query.

In Aurora DSQL, non è possibile eseguire il comando `ANALYZE` all'interno di una transazione esplicita. `ANALYZE`non è soggetto al limite di timeout delle transazioni del database.

Per ridurre la necessità di interventi manuali e mantenere le statistiche costantemente aggiornate, Aurora DSQL esegue automaticamente `ANALYZE` come processo in background. Questo processo in background viene attivato automaticamente in base al tasso di variazione osservato nella tabella. È collegato al numero di righe (tuple) che sono state inserite, aggiornate o eliminate dall'ultima analisi.

Il processo `ANALYZE` viene eseguito in modo asincrono in background e la relativa attività può essere monitorata nella vista di sistema sys.jobs con la seguente query:

```
SELECT * FROM sys.jobs WHERE job_type = 'ANALYZE';
```

Considerazioni chiave

> **Note**
>
> I processi ANALYZE vengono fatturati come gli altri processi asincroni in Aurora DSQL. Quando modifichi una tabella, ciò può attivare indirettamente un processo automatico di raccolta di statistiche in background, che può comportare addebiti di costi dovuti all'attività associata a livello di sistema.

I processi ANALYZE in background, attivati automaticamente, raccolgono gli stessi tipi di statistiche utilizzate da un processo ANALYZE avviato manualmente e le applicano per impostazione predefinita alle tabelle utente. Le tabelle di sistema e di catalogo sono escluse da questo processo automatizzato.

Utilizzo dei piani Aurora DSQL EXPLAIN

Aurora DSQL utilizza una struttura del piano EXPLAIN simile a PostgreSQL, ma con aggiunte chiave che riflettono l'architettura distribuita e il modello di esecuzione.

In questa documentazione, forniremo una panoramica dei piani di Aurora DSQL EXPLAIN, evidenziando le somiglianze e le differenze rispetto a PostgreSQL. Tratteremo i vari tipi di operazioni di scansione disponibili in Aurora DSQL e ti aiuteremo a comprendere i costi di esecuzione delle tue query.

Piani EXPLAIN di PostgreSQL VS Aurora DSQL

Aurora DSQL si basa sul database PostgreSQL e condivide la maggior parte delle strutture del piano con PostgreSQL, ma presenta differenze architettoniche chiave che influiscono sull'esecuzione e l'ottimizzazione delle query:

Funzionalità	PostgreSQL	Aurora DSQL
Storage dei dati	Archiviazione Heap	Nessun heap, tutte le righe sono indicizzate da un identificatore univoco
Chiave primaria	L'indice della chiave primaria è separato dai dati della tabella	L'indice della chiave primaria è la tabella con tutte le colonne

Funzionalità	PostgreSQL	Aurora DSQL
		aggiuntive come colonne INCLUDE
Indici secondari	Indici secondari standard	Funziona come PostgreSQL, con la possibilità di includere colonne non chiave
Funzionalità di filtraggio	Condizione dell'indice, filtro Heap	Condizione dell'indice, filtro di archiviazione, filtro del processore di query
Tipi di scansione	Scansione sequenziale, scansione dell'indice, scansione solo dell'indice	Scansione completa, scansione solo indice, scansione indice
Esecuzione della query	Locale al database	Distribuito (elaborazione e archiviazione sono separati)

Aurora DSQL archivia i dati della tabella direttamente nell'ordine della chiave primaria anziché in un heap separato. Ogni riga è identificata da una chiave univoca, in genere la chiave primaria, che consente al database di ottimizzare le ricerche in modo più efficiente. La differenza architetturale spiega perché Aurora DSQL utilizza spesso le scansioni Index Only nei casi in cui PostgreSQL potrebbe scegliere una scansione sequenziale.

Un'altra differenza fondamentale è che Aurora DSQL separa l'elaborazione dallo storage, consentendo l'applicazione di filtri nelle fasi iniziali del percorso di esecuzione per ridurre lo spostamento dei dati e migliorare le prestazioni.

Per ulteriori informazioni sull'utilizzo dei piani EXPLAIN con PostgreSQL, consulta la documentazione di PostgreSQL EXPLAIN.

Elementi chiave dei piani Aurora DSQL EXPLAIN

I piani Aurora DSQL EXPLAIN forniscono informazioni dettagliate su come vengono eseguite le query, incluso dove avviene il filtraggio e quali colonne vengono recuperate dallo storage. La comprensione di questo risultato consente di ottimizzare le prestazioni delle query.

Index Cond

Condizioni utilizzate per navigare nell'indice. Filtraggio più efficiente che riduce i dati scansionati. In Aurora DSQL, le condizioni dell'indice possono essere applicate a più livelli del piano di esecuzione.

Proiezioni

Colonne recuperate dalla memorizzazione. Un numero inferiore di proiezioni significa prestazioni migliori.

Filtro di archiviazione

Condizioni applicate a livello di archiviazione. Più efficiente dei filtri del processore di query.

Filtro del processore di query

Condizioni applicate a livello di processore di query. Richiede il trasferimento di tutti i dati prima del filtraggio, il che comporta un maggiore spostamento dei dati e un sovraccarico di elaborazione.

Filtri in Aurora DSQL

Aurora DSQL separa l'elaborazione dallo storage, il che significa che il punto in cui vengono applicati i filtri durante l'esecuzione delle query ha un impatto significativo sulle prestazioni. I filtri applicati prima del trasferimento di grandi volumi di dati riducono la latenza e migliorano l'efficienza. Quanto prima viene applicato un filtro, tanto meno dati devono essere elaborati, spostati e scansionati, con conseguenti query più rapide.

Aurora DSQL può applicare filtri in più fasi del percorso della query. La comprensione di queste fasi è fondamentale per interpretare i piani di interrogazione e ottimizzare le prestazioni.

Livello	Tipo di filtro	Description
1	Condizione dell'indice	Applicata durante la scansione dell'indice. Limita la quantità di dati letti dallo storage e riduce i dati inviati al livello di elaborazione.
2	Filtro di archiviazione	Applicato dopo la lettura dei dati dallo storage ma prima di essere inviati al calcolo. Un esempio qui è un filter su una colonna di inclusione di un indice.

Livello	Tipo di filtro	Description
		Riduce il trasferimento di dati ma non la quantità letta.
3	Filtro Query Processor	Applicato dopo che i dati raggiungono il livello di elaborazione. Tutti i dati devono essere trasferiti per primi, il che aumenta la latenza e i costi. Attualmente, Aurora DSQL non è in grado di eseguire tutte le operazioni di filtraggio e proiezione sullo storage, quindi alcune query potrebbero essere costrette a ricorrere a questo tipo di filtro.

Leggere i piani di Aurora DSQL EXPLAIN

Capire come leggere i piani EXPLAIN è fondamentale per ottimizzare le prestazioni delle query. In questa sezione, esamineremo esempi reali di piani di query DSQL di Aurora, mostreremo come si comportano i diversi tipi di scansione, spiegheremo dove vengono applicati i filtri ed evidenzieremo le opportunità di ottimizzazione.

Esempio di scansione completa

Aurora DSQL dispone sia di scansioni sequenziali, identiche dal punto di vista funzionale a PostgreSQL, sia di scansioni complete. L'unica differenza tra queste due è che le scansioni complete possono utilizzare filtri aggiuntivi sullo storage. Per questo motivo, viene quasi sempre selezionato al di sopra delle scansioni sequenziali. A causa della somiglianza, tratteremo solo esempi delle scansioni complete più interessanti.

Le scansioni complete verranno utilizzate principalmente su tabelle prive di chiave primaria. Poiché le chiavi primarie Aurora DSQL sono per impostazione predefinita indici a copertura completa, Aurora DSQL utilizzerà molto probabilmente Index Only Scans sulla chiave primaria in molte situazioni in cui PostgreSQL utilizzerebbe una scansione sequenziale. Come con la maggior parte degli altri database, una tabella senza indici si ridimensionerà male.

```
EXPLAIN SELECT account_id FROM transaction WHERE transaction_date > '2025-01-01' AND
 description LIKE '%external%';
```

```
                                                QUERY PLAN
```

```
------------------------------------------------------------------------------------------
 Full Scan (btree-table) on transaction  (cost=125100.05..177933.38 rows=33333
 width=16)
   Filter: (description ~~ '%external%'::text)
   -> Storage Scan on transaction (cost=12510.05..17793.38 rows=66666 width=16)
        Projections: account_id, description
        Filters: (transaction_date > '2025-01-01 00:00:00'::timestamp without time
 zone)
        -> B-Tree Scan on transaction (cost=12510.05..17793.38 rows=100000 width=30)
```

Questo piano mostra due filtri applicati in fasi diverse. La `transaction_date > '2025-01-01'` condizione viene applicata a livello di archiviazione, riducendo la quantità di dati restituiti. La `description LIKE '%external%'` condizione viene applicata successivamente nel processore di query, dopo il trasferimento dei dati, rendendolo meno efficiente. L'inserimento di filtri più selettivi nei livelli di archiviazione o di indice generalmente migliora le prestazioni.

Esempio di scansione Index Only

Le scansioni indicizzate sono i tipi di scansione più ottimali in Aurora DSQL in quanto comportano il minor numero di round trip verso il livello di archiviazione e possono eseguire la maggior parte dei filtri. Ma solo perché vedi Index Only Scan non significa che tu abbia il piano migliore. A causa di tutti i diversi livelli di filtraggio che possono verificarsi, è essenziale prestare comunque attenzione ai diversi luoghi in cui può avvenire il filtraggio.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending';
```

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Only Scan using idx1 on account  (cost=725.05..1025.08 rows=8 width=18)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   Filter: (balance > '100'::numeric)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=9 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on idx1 (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
```

In questo piano, la condizione dell'indice,`customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'`), viene valutata innanzitutto durante la scansione dell'indice, che è la fase più efficiente perché limita la quantità di dati letti dallo storage. Il filtro di archiviazione,`status = 'pending'`, viene applicato dopo la lettura dei dati ma prima di essere inviati al livello di elaborazione, riducendo la quantità di dati trasferiti. Infine, il filtro del processore di query viene eseguito per ultimo, dopo lo spostamento dei dati, il che lo rende il meno efficiente. `balance > 100` Di queste, la condizione dell'indice offre le prestazioni migliori perché controlla direttamente la quantità di dati scansionati.

Esempio di Index Scan

Le scansioni degli indici sono simili alle scansioni di solo indice, tranne per il fatto che hanno il passaggio aggiuntivo di dover richiamare la tabella di base. Poiché Aurora DSQL è in grado di specificare filtri di archiviazione, è in grado di farlo sia sulla chiamata di indice che sulla chiamata di ricerca.

Per chiarire questo punto, Aurora DSQL presenta il piano come due nodi. In questo modo, puoi vedere chiaramente quanto l'aggiunta di una colonna di inclusione possa aiutare in termini di righe restituite dall'archiviazione.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending'
AND created_at > '2025-01-01';
```

```
                                                QUERY PLAN
-------------------------------------------------------------------------------------------------
 Index Scan using idx1 on account  (cost=728.18..1132.20 rows=3 width=18)
   Filter: (balance > '100'::numeric)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=8 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on account (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Lookup on account (cost=12510.05..17793.38 rows=4 width=16)
        Filters: (created_at > '2025-01-01 00:00:00'::timestamp without time zone)
        -> B-Tree Lookup on transaction (cost=12510.05..17793.38 rows=8 width=30)
```

Questo piano mostra come avviene il filtraggio in più fasi:

- La condizione di indicizzazione `customer_id` filtra i dati in anticipo.
- Il filtro di archiviazione limita `status` ulteriormente i risultati prima che vengano inviati al calcolo.
- Il filtro del processore di query attivo `balance` viene applicato successivamente, dopo il trasferimento.
- Il filtro di ricerca attivato `created_at` viene valutato quando si recuperano colonne aggiuntive dalla tabella di base.

L'aggiunta di colonne utilizzate di frequente come `INCLUDE` campi può spesso eliminare questa ricerca e migliorare le prestazioni.

Best practice

- Allinea i filtri alle colonne indicizzate per accelerare il filtraggio.
- Utilizza le colonne INCLUDE per consentire le scansioni solo indicizzate ed evitare le ricerche.
- Mantieni aggiornate le statistiche per garantire che le stime dei costi e delle righe siano accurate.
- Evita le query non indicizzate su tabelle di grandi dimensioni per evitare costose scansioni complete.

Comprensione DPUs in EXPLAIN ANAL

Aurora DSQL fornisce informazioni DPU (Distributed Processing Unit) a livello di dichiarazione nell'output del `EXPLAIN ANALYZE VERBOSE` piano, offrendoti una visibilità più approfondita sui costi delle query durante lo sviluppo. Questa sezione spiega cosa DPUs sono e come interpretarli nell'output. `EXPLAIN ANALYZE VERBOSE`

Che cos'è una DPU?

Un'unità di elaborazione distribuita (DPU) è la misura normalizzata del lavoro svolto da Aurora DSQL. È composta da:

- ComputedPU: tempo impiegato per l'esecuzione di query SQL
- ReadDPU: risorse utilizzate per leggere i dati dallo storage
- WriteDPU: risorse utilizzate per scrivere dati nell'archivio

- MultiRegionWriteDPU: risorse utilizzate per replicare le scritture su cluster peer in configurazioni multiregionali.

Utilizzo della DPU in EXPLAIN ANALYZE VERBOSE

Aurora DSQL si estende `EXPLAIN ANALYZE VERBOSE` per includere una stima dell'utilizzo della DPU a livello di dichiarazione fino alla fine dell'output. Ciò offre una visibilità immediata sui costi delle query, aiutandoti a identificare i fattori di costo del carico di lavoro, ottimizzare le prestazioni delle query e prevedere meglio l'utilizzo delle risorse.

Gli esempi seguenti mostrano come interpretare le stime della DPU a livello di dichiarazione incluse nell'output EXPLAIN ANALYZE VERBOSE.

Esempio 1: SELECT Query

```
EXPLAIN ANALYZE VERBOSE SELECT * FROM test_table;
```

```
QUERY PLAN
-------------------------------------------------
Index Only Scan using test_table_pkey on public.test_table  (cost=125100.05..171100.05
 rows=1000000 width=36) (actual time=2.973..4.482 rows=120 loops=1)
  Output: id, context
  -> Storage Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000 width=36)
 (actual rows=120 loops=1)
      Projections: id, context
      -> B-Tree Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000
 width=36) (actual rows=120 loops=1)
Query Identifier: qymgw1m77maoe
Planning Time: 11.415 ms
Execution Time: 4.528 ms
Statement DPU Estimate:
  Compute: 0.01607 DPU
  Read: 0.04312 DPU
  Write: 0.00000 DPU
  Total: 0.05919 DPU
```

In questo esempio, l'istruzione SELECT esegue una scansione solo indicizzata, quindi la maggior parte del costo proviene da Read DPU (0.04312), che rappresenta i dati recuperati dallo storage e Compute DPU (0,01607), che riflette le risorse di calcolo utilizzate per elaborare e restituire i risultati.

Non esiste una DPU di scrittura poiché la query non modifica i dati. La DPU totale (0,05919) è la somma di Compute+Read+Write.

Esempio 2: INSERT Query

```
EXPLAIN ANALYZE VERBOSE INSERT INTO test_table VALUES (1, 'name1'), (2, 'name2'), (3,
 'name3');
```

```
QUERY PLAN
-----------------------------------------------------
Insert on public.test_table  (cost=0.00..0.04 rows=0 width=0) (actual time=0.055..0.056
 rows=0 loops=1)
  ->  Values Scan on "*VALUES*"  (cost=0.00..0.04 rows=3 width=122) (actual
 time=0.003..0.008 rows=3 loops=1)
       Output: "*VALUES*".column1, "*VALUES*".column2
Query Identifier: jtkjkexhjotbo
Planning Time: 0.068 ms
Execution Time: 0.543 ms
Statement DPU Estimate:
  Compute: 0.01550 DPU
  Read: 0.00307 DPU (Transaction minimum: 0.00375)
  Write: 0.01875 DPU (Transaction minimum: 0.05000)
  Total: 0.03732 DPU
```

Questa istruzione esegue principalmente le scritture, quindi la maggior parte del costo è associato a Write DPU. La Compute DPU (0,01550) rappresenta il lavoro svolto per elaborare e inserire i valori. La Read DPU (0.00307) riflette le letture secondarie del sistema (per le ricerche nel catalogo o il controllo degli indici).

Notate i valori minimi delle transazioni mostrati accanto a Lettura e scrittura. DPUs Questi indicano i costi di base per transazione che si applicano solo quando l'operazione include operazioni di lettura o scrittura. Ciò non significa che ogni transazione comporti automaticamente un costo di 0,00375 DPU in lettura o 0,05 DPU in scrittura. Questi minimi vengono invece applicati a livello di transazione durante l'aggregazione dei costi e solo se all'interno della transazione vengono eseguite operazioni di lettura o scrittura. A causa di questa differenza di ambito, le stime a livello di rendiconto `EXPLAIN ANALYZE VERBOSE` potrebbero non corrispondere esattamente alle metriche a livello di transazione riportate nei dati di fatturazione o nei dati di fatturazione. CloudWatch

Utilizzo delle informazioni sulla DPU per l'ottimizzazione

Le stime della DPU per dichiarazione offrono un modo efficace per ottimizzare le query oltre il semplice tempo di esecuzione. Casi di utilizzo comune comprendono:

- Consapevolezza dei costi: scopri quanto è costosa una query rispetto alle altre.
- Ottimizzazione dello schema: confronta l'impatto degli indici o delle modifiche allo schema sulle prestazioni e sull'efficienza delle risorse.
- Pianificazione del budget: stima del costo del carico di lavoro in base all'utilizzo della DPU osservato.
- Confronto tra query: valuta gli approcci di interrogazione alternativi in base al consumo relativo di DPU.

Interpretazione delle informazioni sulla DPU

Tieni a mente le seguenti best practice quando utilizzi dati DPU da: `EXPLAIN ANALYZE VERBOSE`

- Usalo in modo direzionale: considera la DPU segnalata come un modo per comprendere il costo relativo di una query piuttosto che come una corrispondenza esatta con le CloudWatch metriche o i dati di fatturazione. Le differenze sono previste perché `EXPLAIN ANALYZE VERBOSE` riporta i costi a livello di rendiconto, mentre aggrega l'attività a livello di transazione. CloudWatch CloudWatch include anche operazioni in background (come ANALYZE o compactions) e il sovraccarico delle transazioni (/) che esclude intenzionalmente. `BEGIN` `COMMIT` `EXPLAIN ANALYZE VERBOSE`
- La variabilità della DPU tra le esecuzioni è normale nei sistemi distribuiti e non indica errori. Fattori come la memorizzazione nella cache, le modifiche al piano di esecuzione, la concorrenza o i cambiamenti nella distribuzione dei dati possono tutti far sì che la stessa query utilizzi risorse diverse da un'esecuzione all'altra.
- Operazioni in batch di piccole dimensioni: se il carico di lavoro genera molte istruzioni di piccole dimensioni, valuta la possibilità di raggrupparle in batch in operazioni più grandi (non superare i 10 MB). In questo modo si riducono le spese generali di arrotondamento e si ottengono stime dei costi più significative.
- Utilizzabile per l'ottimizzazione, non per la fatturazione: i dati in `EXPLAIN ANALYZE VERBOSE` ingresso della DPU sono progettati per la consapevolezza dei costi, l'ottimizzazione delle query e l'ottimizzazione. Non è una metrica adatta alla fatturazione. Affidati sempre alle CloudWatch metriche o ai rapporti di fatturazione mensili per dati autorevoli su costi e utilizzo.

Gestione di cluster Aurora DSQL

Aurora DSQL offre diverse opzioni di configurazione da utilizzare per stabilire l'infrastruttura di database giusta per ogni esigenza. Per configurare l'infrastruttura del cluster Aurora DSQL, consulta le seguenti sezioni.

Argomenti

Le caratteristiche e le funzionalità illustrate in questa guida assicurano che l'ambiente Aurora DSQL sia più resiliente, reattivo e in grado di supportare le applicazioni man mano che crescono ed evolvono.

Configurazione di cluster a Regione singola

Configurazione e gestione dei cluster di una Regione AWS utilizzando la AWS CLI o il linguaggio di programmazione preferito, tra cui Python, C++, JavaScript, Java, Rust, Ruby, .NET e Golang. La AWS CLI fornisce un accesso rapido tramite i comandi della shell, mentre i Software Development Kit (SDK) AWS consentono il controllo programmatico tramite il supporto del linguaggio nativo.

Argomenti

Utilizzo degli SDK AWS

Gli SDK AWS forniscono l'accesso programmatico ad Aurora DSQL nel linguaggio di programmazione preferito. Le sezioni seguenti mostrano come eseguire operazioni comuni sui cluster utilizzando diversi linguaggi di programmazione.

Creazione di un cluster

Negli esempi seguenti viene illustrato come creare un cluster a Regione singola utilizzando diversi linguaggi di programmazione.

Python

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
import boto3


def create_cluster(region):
    try:
        client = boto3.client("dsql", region_name=region)
        tags = {"Name": "Python single region cluster"}
        cluster = client.create_cluster(tags=tags, deletionProtectionEnabled=True)
        print(f"Initiated creation of cluster: {cluster["identifier"]}")

        print(f"Waiting for {cluster["arn"]} to become ACTIVE")
        client.get_waiter("cluster_active").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        return cluster
    except:
        print("Unable to create cluster")
        raise


def main():
    region = "us-east-1"
    response = create_cluster(region)
    print(f"Created cluster: {response["arn"]}")


if __name__ == "__main__":
    main()
```

C++

L'esempio seguente consente di creare un cluster in un'unica Regione AWS.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates a single-region cluster in Amazon Aurora DSQL
 */
CreateClusterResult CreateCluster(const Aws::String& region) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create the cluster
    CreateClusterRequest createClusterRequest;
    createClusterRequest.SetDeletionProtectionEnabled(true);
    createClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp single region cluster";
    createClusterRequest.SetTags(tags);

    auto createOutcome = client.CreateCluster(createClusterRequest);
    if (!createOutcome.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region << ": "
                  << createOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to create cluster in " + region);
    }

    auto cluster = createOutcome.GetResult();
```

```
    std::cout << "Created " << cluster.GetArn() << std::endl;

    return cluster;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region for the single-region setup
            Aws::String region = "us-east-1";

            auto cluster = CreateCluster(region);

            std::cout << "Created single region cluster:" << std::endl;
            std::cout << "Cluster ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Cluster Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
import { DSQLClient, CreateClusterCommand, waitUntilClusterActive } from "@aws-sdk/
client-dsql";

async function createCluster(region) {

    const client = new DSQLClient({ region });

    try {
        const createClusterCommand = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript single region cluster"
```

```
            },
        });
        const response = await client.send(createClusterCommand);

        console.log(`Waiting for cluster ${response.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response.identifier
            }
        );
        console.log(`Cluster Id ${response.identifier} is now active`);
        return;
    } catch (error) {
        console.error(`Unable to create cluster in ${region}: `, error.message);
        throw error;
    }
}

async function main() {
    const region = "us-east-1";

    await createCluster(region);
}

main();
```

Java

Usa l'esempio seguente per creare un cluster in una singola Regione AWS.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
```

```
import java.time.Duration;
import java.util.Map;

public class CreateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            CreateClusterRequest request = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .tags(Map.of("Name", "java single region cluster"))
                    .build();
            CreateClusterResponse cluster = client.createCluster(request);
            System.out.println("Created " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for a cluster's
            // transition to ACTIVE.
            System.out.println("Waiting for cluster to become ACTIVE");
            WaiterResponse<GetClusterResponse> waiterResponse =
 client.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster.identifier()),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            waiterResponse.matched().response().ifPresent(System.out::println);
        }
    }
}
```

Rust

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
```

```
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    let region_provider = Region::new(region);

    let config = load_defaults(BehaviorVersion::latest())
        .region(region_provider)
        .load()
        .await;

    let config = Config::new(&config);

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;

    let tags = HashMap::from([
        (String::from("Name"), String::from("rust single region cluster")),
    ]);

    println!("Creating cluster in {region}");
    let cluster = client
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .send()
        .await
        .unwrap();

    println!("Created {}", cluster.arn);

    println!("Waiting for {} to become ACTIVE", cluster.arn);
    let cluster_output = client
        .wait_until_cluster_active()
        .identifier(&cluster.identifier)
        .send()
```

```
        .await
        .unwrap();

    cluster_output
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let region = "us-east-1";

    let cluster = create_cluster(region).await;

    println!("Created single region cluster:");
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
require "aws-sdk-dsql"
require "pp"

def create_cluster(region)
  client = Aws::DSQL::Client.new(region: region)

  puts "Creating cluster in #{region}"
  cluster = client.create_cluster(
    deletion_protection_enabled: true,
    tags: {
      Name: "ruby single region cluster"
    }
  )
  puts "Created #{cluster.arn}"

  puts "Waiting for #{cluster.arn} to become ACTIVE"
  cluster = client.wait_until(:cluster_active, identifier: cluster.identifier) do |
w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
  cluster
rescue Aws::Errors::ServiceError => e
  abort "Failed to create cluster: #{e.message}"
end

def main
  region = "us-east-1"

  cluster = create_cluster(region)

  puts "Created single region cluster:"
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class CreateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = new DefaultAWSCredentialsChain().GetCredentials();
            var clientConfig = new AmazonDSQLConfig
```

```
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create a cluster with deletion protection enabled and a name tag.
        /// </summary>
        public static async Task<CreateClusterResponse> Create(RegionEndpoint
 region)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp single region cluster" }
                };

                var createClusterRequest = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags
                };

                var cluster = await client.CreateClusterAsync(createClusterRequest);
                Console.WriteLine($"Created {cluster.Arn}");

                return cluster;
            }
        }

        public static async Task Main()
        {
            var region = RegionEndpoint.USEast1;

            var cluster = await Create(region);

            Console.WriteLine("Created single region cluster:");
            Console.WriteLine($"Cluster ARN: {cluster.Arn}");
        }
    }
}
```

Golang

Per creare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func CreateCluster(ctx context.Context, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

 input := &dsql.CreateClusterInput{
  DeletionProtectionEnabled: &deleteProtect,
  Tags: map[string]string{
   "Name": "go single-region cluster",
  },
 }

 clusterProperties, err := client.CreateCluster(context.Background(), input)

 if err != nil {
  return fmt.Errorf("failed to create cluster. %v", err)
 }

 // Create the waiter with our custom options
 waiter := dsql.NewClusterActiveWaiter(client, func(o
 *dsql.ClusterActiveWaiterOptions) {
```

```
  o.MaxDelay = 30 * time.Second
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })

 // Create the input for the clusterProperties to monitor
 clusterInput := &dsql.GetClusterInput{
  Identifier: clusterProperties.Identifier,
 }

 fmt.Printf("Waiting for cluster %s to become ACTIVE\n", *clusterProperties.Arn)
 err = waiter.Wait(ctx, clusterInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to become active: %w", err)
 }

 fmt.Printf("Created single region cluster: %s\n", *clusterProperties.Arn)
 return nil
}

func main() {
 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := CreateCluster(ctx, "us-east-1")
 if err != nil {
  fmt.Printf("failed to create cluster: %v", err)
  panic(err)
 }

}
```

Recupero delle informazioni di un cluster

Negli esempi seguenti viene illustrato come ottenere informazioni su un cluster a Regione singola utilizzando diversi linguaggi di programmazione.

Python

Per ottenere informazioni su un cluster a Regione singola, utilizzare l'esempio seguente.

```
import boto3
from datetime import datetime
import json


def get_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.get_cluster(identifier=identifier)
    except:
        print(f"Unable to get cluster {identifier} in region {region}")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    response = get_cluster(region, cluster_id)

    print(json.dumps(response, indent=2, default=lambda obj: obj.isoformat() if
 isinstance(obj, datetime) else None))


if __name__ == "__main__":
    main()
```

C++

Utilizzare l'esempio seguente per ottenere informazioni su un cluster a Regione singola.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Retrieves information about a cluster in Amazon Aurora DSQL
```

```
 */
GetClusterResult GetCluster(const Aws::String& region, const Aws::String&
 identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Get the cluster
    GetClusterRequest getClusterRequest;
    getClusterRequest.SetIdentifier(identifier);

    auto getOutcome = client.GetCluster(getClusterRequest);
    if (!getOutcome.IsSuccess()) {
        std::cerr << "Failed to retrieve cluster " << identifier << " in " << region
 << ": "
                  << getOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to retrieve cluster " + identifier + " in
 region " + region);
    }

    return getOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            auto cluster = GetCluster(region, clusterId);

            // Print cluster details
            std::cout << "Cluster Details:" << std::endl;
            std::cout << "ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
```

```
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per ottenere informazioni su un cluster a Regione singola, utilizzare l'esempio seguente.

```
import { DSQLClient, GetClusterCommand } from "@aws-sdk/client-dsql";

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  const getClusterCommand = new GetClusterCommand({
    identifier: clusterId,
  });

  try {
    return await client.send(getClusterCommand);
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or deleted");
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  const response = await getCluster(region, clusterId);
  console.log("Cluster: ", response);
}

main();
```

Java

L'esempio seguente consente di ottenere informazioni su un cluster a Regione singola.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

public class GetCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            GetClusterResponse cluster = client.getCluster(r ->
 r.identifier(clusterId));
            System.out.println(cluster);
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

L'esempio seguente consente di ottenere informazioni su un cluster a Regione singola.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
```

```
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Get a ClusterResource from DSQL cluster identifier
pub async fn get_cluster(region: &'static str, identifier: &'static str) ->
 GetClusterOutput {
    let client = dsql_client(region).await;
    client
        .get_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = get_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

L'esempio seguente consente di ottenere informazioni su un cluster a Regione singola.

```
require "aws-sdk-dsql"
require "pp"

def get_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  client.get_cluster(identifier: identifier)
rescue Aws::Errors::ServiceError => e
  abort "Unable to retrieve cluster #{identifier} in region #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  cluster = get_cluster(region, cluster_id)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

L'esempio seguente consente di ottenere informazioni su un cluster a Regione singola.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class GetCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
```

```
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Get information about a DSQL cluster.
        /// </summary>
        public static async Task<GetClusterResponse> Get(RegionEndpoint region,
 string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var getClusterRequest = new GetClusterRequest
                {
                    Identifier = identifier
                };

                return await client.GetClusterAsync(getClusterRequest);
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            var response = await Get(region, clusterId);
            Console.WriteLine($"Cluster ARN: {response.Arn}");
        }
    }
}
```

Golang

L'esempio seguente consente di ottenere informazioni su un cluster a Regione singola.

```
package main
```

```
import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func GetCluster(ctx context.Context, region, identifier string) (clusterStatus
 *dsql.GetClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := &dsql.GetClusterInput{
  Identifier: aws.String(identifier),
 }
 clusterStatus, err = client.GetCluster(context.Background(), input)

 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }

 log.Printf("Cluster ARN: %s", *clusterStatus.Arn)

 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 _, err := GetCluster(ctx, region, identifier)
```

```
 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }
}
```

Aggiornamento di un cluster

Gli esempi seguenti mostrano come aggiornare un cluster a Regione singola utilizzando diversi linguaggi di programmazione.

Python

Per aggiornare un cluster a Regione singola, utilizzare l'esempio seguente.

```
import boto3


def update_cluster(region, cluster_id, deletion_protection_enabled):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.update_cluster(identifier=cluster_id,
 deletionProtectionEnabled=deletion_protection_enabled)
    except:
        print("Unable to update cluster")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    deletion_protection_enabled = False
    response = update_cluster(region, cluster_id, deletion_protection_enabled)
    print(f"Updated {response["arn"]} with deletion_protection_enabled:
 {deletion_protection_enabled}")


if __name__ == "__main__":
    main()
```

C++

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Updates a cluster in Amazon Aurora DSQL
 */
UpdateClusterResult UpdateCluster(const Aws::String& region, const
 Aws::Map<Aws::String, Aws::String>& updateParams) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create update request
    UpdateClusterRequest updateRequest;
    updateRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Set identifier (required)
    if (updateParams.find("identifier") != updateParams.end()) {
        updateRequest.SetIdentifier(updateParams.at("identifier"));
    } else {
        throw std::runtime_error("Cluster identifier is required for update
 operation");
    }

    // Set deletion protection if specified
    if (updateParams.find("deletion_protection_enabled") != updateParams.end()) {
        bool deletionProtection = (updateParams.at("deletion_protection_enabled") ==
 "true");
        updateRequest.SetDeletionProtectionEnabled(deletionProtection);
    }

    // Execute the update
```

```
    auto updateOutcome = client.UpdateCluster(updateRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster: " <<
 updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to update cluster");
    }

    return updateOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and update parameters
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            // Create parameter map
            Aws::Map<Aws::String, Aws::String> updateParams;
            updateParams["identifier"] = clusterId;
            updateParams["deletion_protection_enabled"] = "false";

            auto updatedCluster = UpdateCluster(region, updateParams);

            std::cout << "Updated " << updatedCluster.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per aggiornare un cluster a Regione singola, utilizzare l'esempio seguente.

```
import { DSQLClient, UpdateClusterCommand } from "@aws-sdk/client-dsql";

export async function updateCluster(region, clusterId, deletionProtectionEnabled) {
```

```
  const client = new DSQLClient({ region });

  const updateClusterCommand = new UpdateClusterCommand({
    identifier: clusterId,
    deletionProtectionEnabled: deletionProtectionEnabled
  });

  try {
    return await client.send(updateClusterCommand);
  } catch (error) {
    console.error("Unable to update cluster", error.message);
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";
  const deletionProtectionEnabled = false;

  const response = await updateCluster(region, clusterId,
 deletionProtectionEnabled);
  console.log(`Updated ${response.arn}`);
}

main();
```

Java

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;
import software.amazon.awssdk.services.dsql.model.UpdateClusterResponse;

public class UpdateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
```

```
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            UpdateClusterRequest request = UpdateClusterRequest.builder()
                    .identifier(clusterId)
                    .deletionProtectionEnabled(false)
                    .build();
            UpdateClusterResponse cluster = client.updateCluster(request);
            System.out.println("Updated " + cluster.arn());
        }
    }
}
```

Rust

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::update_cluster::UpdateClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();
```

```
    Client::from_conf(config)
}

/// Update a DSQL cluster and set delete protection to false. Also add new tags.
pub async fn update_cluster(region: &'static str, identifier: &'static str) ->
 UpdateClusterOutput {
    let client = dsql_client(region).await;
    // Update delete protection
    let update_response = client
        .update_cluster()
        .identifier(identifier)
        .deletion_protection_enabled(false)
        .send()
        .await
        .unwrap();

    update_response
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = update_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
require "aws-sdk-dsql"

def update_cluster(region, update_params)
  client = Aws::DSQL::Client.new(region: region)
  client.update_cluster(update_params)
rescue Aws::Errors::ServiceError => e
  abort "Unable to update cluster: #{e.message}"
end

def main
```

```
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  updated_cluster = update_cluster(region, {
    identifier: cluster_id,
    deletion_protection_enabled: false
  })
  puts "Updated #{updated_cluster.arn}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class UpdateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
```

```
        /// Update a DSQL cluster and set delete protection to false.
        /// </summary>
        public static async Task<UpdateClusterResponse> Update(RegionEndpoint
 region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = identifier,
                    DeletionProtectionEnabled = false
                };

                UpdateClusterResponse response = await
 client.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Updated {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            await Update(region, clusterId);
        }
    }
}
```

Golang

Utilizzare l'esempio seguente per aggiornare un cluster a Regione singola.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
```

```
)

func UpdateCluster(ctx context.Context, region, id string, deleteProtection bool)
 (clusterStatus *dsql.UpdateClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := dsql.UpdateClusterInput{
  Identifier:                &id,
  DeletionProtectionEnabled: &deleteProtection,
 }

 clusterStatus, err = client.UpdateCluster(context.Background(), &input)

 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }

 log.Printf("Cluster updated successfully: %v", clusterStatus.Status)
 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"
 deleteProtection := false

 _, err := UpdateCluster(ctx, region, identifier, deleteProtection)
 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }
}
```

Eliminazione di un cluster

Gli esempi seguenti mostrano come cancellare un cluster a Regione singola utilizzando diversi linguaggi di programmazione.

Python

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
import boto3


def delete_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        cluster = client.delete_cluster(identifier=identifier)
        print(f"Initiated delete of {cluster["arn"]}")

        print("Waiting for cluster to finish deletion")
        client.get_waiter("cluster_not_exists").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )
    except:
        print("Unable to delete cluster " + identifier)
        raise


def main():
    region = "us-east-1"
    cluster_id = "<cluster id>"  # Use a placeholder in docs
    delete_cluster(region, cluster_id)
    print(f"Deleted {cluster_id}")


if __name__ == "__main__":
    main()
```

C++

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes a single-region cluster in Amazon Aurora DSQL
 */
void DeleteCluster(const Aws::String& region, const Aws::String& identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Delete the cluster
    DeleteClusterRequest deleteRequest;
    deleteRequest.SetIdentifier(identifier);
    deleteRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome = client.DeleteCluster(deleteRequest);
    if (!deleteOutcome.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << identifier << " in " << region
 << ": "
                  << deleteOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to delete cluster " + identifier + " in " +
 region);
    }

    auto cluster = deleteOutcome.GetResult();
    std::cout << "Initiated delete of " << cluster.GetArn() << std::endl;
}
```

```
int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            DeleteCluster(region, clusterId);

            std::cout << "Deleted " << clusterId << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  try {
    const deleteClusterCommand = new DeleteClusterCommand({
      identifier: clusterId,
    });
    const response = await client.send(deleteClusterCommand);

    console.log(`Waiting for cluster ${response.identifier} to finish deletion`);

    await waitUntilClusterNotExists(
      {
        client: client,
        maxWaitTime: 300 // Wait for 5 minutes
```

```
      },
      {
        identifier: response.identifier
      }
    );
    console.log(`Cluster Id ${response.identifier} is now deleted`);
    return;
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or already deleted");
    } else {
      console.error("Unable to delete cluster: ", error.message);
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  await deleteCluster(region, clusterId);
}

main();
```

Java

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

import java.time.Duration;

public class DeleteCluster {

    public static void main(String[] args) {
```

```
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";


        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            DeleteClusterResponse cluster = client.deleteCluster(r ->
 r.identifier(clusterId));
            System.out.println("Initiated delete of " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for deletion.
            System.out.println("Waiting for cluster to finish deletion");
            client.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            System.out.println("Deleted " + cluster.arn());
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
```

```
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Delete a DSQL cluster
pub async fn delete_cluster(region: &'static str, identifier: &'static str) {
    let client = dsql_client(region).await;
    let delete_response = client
        .delete_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap();
    println!("Initiated delete of {}", delete_response.arn);

    println!("Waiting for cluster to finish deletion");
    client
        .wait_until_cluster_not_exists()
        .identifier(identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";
    let cluster_id = "<cluster to be deleted>";

    delete_cluster(region, cluster_id).await;
    println!("Deleted {cluster_id}");
```

```
    Ok(())
}
```

Ruby

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
require "aws-sdk-dsql"

def delete_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.delete_cluster(identifier: identifier)
  puts "Initiated delete of #{cluster.arn}"

  # The DSQL SDK offers built-in waiters to poll for deletion.
  puts "Waiting for cluster to finish deletion"
  client.wait_until(:cluster_not_exists, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Unable to delete cluster #{identifier} in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  delete_cluster(region, cluster_id)
  puts "Deleted #{cluster_id}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
```

```
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class DeleteSingleRegionCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete a DSQL cluster.
        /// </summary>
        public static async Task Delete(RegionEndpoint region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var deleteRequest = new DeleteClusterRequest
                {
                    Identifier = identifier
                };

                var deleteResponse = await client.DeleteClusterAsync(deleteRequest);
                Console.WriteLine($"Initiated deletion of {deleteResponse.Arn}");
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<cluster to be deleted>";
```

```
            await Delete(region, clusterId);
        }
    }
}
```

Golang

Per eliminare un cluster in una singola Regione AWS, utilizzare l'esempio seguente.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func DeleteSingleRegion(ctx context.Context, identifier, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 // Create delete cluster input
 deleteInput := &dsql.DeleteClusterInput{
  Identifier: &identifier,
 }

 // Delete the cluster
 result, err := client.DeleteCluster(ctx, deleteInput)
 if err != nil {
  return fmt.Errorf("failed to delete cluster: %w", err)
 }

 fmt.Printf("Initiated deletion of cluster: %s\n", *result.Arn)
```

```
 // Create waiter to check cluster deletion
 waiter := dsql.NewClusterNotExistsWaiter(client, func(options
 *dsql.ClusterNotExistsWaiterOptions) {
  options.MinDelay = 10 * time.Second
  options.MaxDelay = 30 * time.Second
  options.LogWaitAttempts = true
 })

 // Create the input for checking cluster status
 getInput := &dsql.GetClusterInput{
  Identifier: &identifier,
 }

 // Wait for the cluster to be deleted
 fmt.Printf("Waiting for cluster %s to be deleted...\n", identifier)
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to be deleted: %w", err)
 }

 fmt.Printf("Cluster %s has been successfully deleted\n", identifier)
 return nil
}

func DeleteCluster(ctx context.Context) {
}

// Example usage in main function
func main() {
 // Your existing setup code for client configuration...

 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 // Need to make sure that cluster does not have delete protection enabled
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 err := DeleteSingleRegion(ctx, identifier, region)
 if err != nil {
  log.Fatalf("Failed to delete cluster: %v", err)
 }
```

```
}
```

Per altri esempi di codice, visitare il repository GitHub degli esempi di Aurora DSQL.

Utilizzo della CLI di AWS

La CLI di AWS fornisce un'interfaccia a riga di comando per la gestione dei cluster Aurora DSQL. L'esempio seguente illustra le operazioni comuni relative alla gestione dei cluster.

Creazione di un cluster

Creazione di un cluster utilizzando il comando create-cluster.

> **Note**
>
> La creazione di cluster è un'operazione asincrona. Invocare l'API GetCluster fino a quando lo stato non cambia a `ACTIVE`. Una volta diventato attivo, è possibile connettersi al cluster.

Example Comando

```
aws dsql create-cluster --region us-east-1
```

> **Note**
>
> Per disabilitare la protezione dall'eliminazione durante la creazione, includere il `--no-deletion-protection-enabled`.

Example Risposta

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "CREATING",
    "creationTime": "2024-05-25T16:56:49.784000-07:00",
    "deletionProtectionEnabled": true,
    "tag": {},
```

```
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

Descrizione di un cluster

Ottenere informazioni su un cluster utilizzando il comando get-cluster.

Example Comando

```
aws dsql get-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Risposta

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "encryptionDetails": {
        "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
        "kmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/123a456b-c789-01de-2f34-
g5hi6j7k8lm9",
        "encryptionStatus": "ENABLED"
    }
}
```

Aggiornamento di un cluster

Aggiornamento di un cluster esistente utilizzando il comando update-cluster.

> **Note**
>
> Gli aggiornamenti sono operazioni asincrone. Per visualizzare le modifiche, invocare l'API GetCluster fino a quando lo stato non cambia a `ACTIVE`.

Example Comando

```
aws dsql update-cluster \
  --region us-east-1 \
  --no-deletion-protection-enabled \
  --identifier your_cluster_id
```

Example Risposta

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

Eliminazione di un cluster

Eliminazione di un cluster esistente utilizzando il comando delete-cluster.

> **Note**
>
> È possibile eliminare solo i cluster per i quali è disabilitata la protezione da eliminazione. Quando si crea un cluster, la protezione dall'eliminazione viene abilitata per impostazione predefinita.

Example Comando

```
aws dsql delete-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Risposta

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
```

```
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

Elencazione dei cluster

Elencazione dei cluster usando il comando list-clusters.

Example Comando

```
aws dsql list-clusters --region us-east-1
```

Example Risposta

```
{
    "clusters": [
        {
            "identifier": "abc0def1baz2quux3quux4quuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuux"
        },
        {
            "identifier": "abc0def1baz2quux3quux5quuuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux5quuuux"
        }
    ]
}
```

Configurazione di cluster multi-Regione

È possibile configurare e gestire cluster distribuiti su più Regioni AWS utilizzando la AWS CLI o il linguaggio di programmazione preferito, tra cui Python, C++, JavaScript, Java, Rust, Ruby, .NET e Golang. La AWS CLI fornisce un accesso rapido tramite i comandi della shell, mentre gli SDK di AWS consentono il controllo programmatico tramite il supporto del linguaggio nativo.

Argomenti

- Utilizzo degli SDK AWS
- Utilizzo della CLI di AWS

Utilizzo degli SDK AWS

Gli SDK AWS forniscono l'accesso programmatico ad Aurora DSQL nel linguaggio di programmazione preferito. Le sezioni seguenti mostrano come eseguire operazioni comuni sui cluster utilizzando diversi linguaggi di programmazione.

Creazione di un cluster

Negli esempi seguenti viene illustrato come creare un cluster multi-Regione utilizzando diversi linguaggi di programmazione.

Python

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
import boto3


def create_multi_region_clusters(region_1, region_2, witness_region):
    try:
        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        # We can only set the witness region for the first cluster
        cluster_1 = client_1.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region},
            tags={"Name": "Python multi region cluster"}
        )
        print(f"Created {cluster_1["arn"]}")

        # For the second cluster we can set witness region and designate cluster_1
 as a peer
        cluster_2 = client_2.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_1["arn"]]},
            tags={"Name": "Python multi region cluster"}
        )

        print(f"Created {cluster_2["arn"]}")
        # Now that we know the cluster_2 arn we can set it as a peer of cluster_1
```

```
        client_1.update_cluster(
            identifier=cluster_1["identifier"],
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_2["arn"]]}
        )
        print(f"Added {cluster_2["arn"]} as a peer of {cluster_1["arn"]}")

        # Now that multiRegionProperties is fully defined for both clusters
        # they'll begin the transition to ACTIVE
        print(f"Waiting for {cluster_1["arn"]} to become ACTIVE")
        client_1.get_waiter("cluster_active").wait(
            identifier=cluster_1["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_2["arn"]} to become ACTIVE")
        client_2.get_waiter("cluster_active").wait(
            identifier=cluster_2["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        return (cluster_1, cluster_2)

    except:
        print("Unable to create cluster")
        raise


def main():
    region_1 = "us-east-1"
    region_2 = "us-east-2"
    witness_region = "us-west-2"
    (cluster_1, cluster_2) = create_multi_region_clusters(region_1, region_2,
 witness_region)
    print("Created multi region clusters:")
    print("Cluster id: " + cluster_1['arn'])
    print("Cluster id: " + cluster_2['arn'])
```

```
if __name__ == "__main__":
    main()
```

C++

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <aws/dsql/model/MultiRegionProperties.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates multi-region clusters in Amazon Aurora DSQL
 */
std::pair<CreateClusterResult, CreateClusterResult> CreateMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& region2,
    const Aws::String& witnessRegion) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    std::cout << "Creating cluster in " << region1 << std::endl;
```

```
    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region
    MultiRegionProperties multiRegionProps1;
    multiRegionProps1.SetWitnessRegion(witnessRegion);
    createClusterRequest1.SetMultiRegionProperties(multiRegionProps1);

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp multi region cluster 1";
    createClusterRequest1.SetTags(tags);
    createClusterRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto createOutcome1 = client1.CreateCluster(createClusterRequest1);
    if (!createOutcome1.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region1 << ": "
                  << createOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster1 = createOutcome1.GetResult();
    std::cout << "Created " << cluster1.GetArn() << std::endl;

    // Create second cluster
    std::cout << "Creating cluster in " << region2 << std::endl;

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region and cluster1 as peer
    MultiRegionProperties multiRegionProps2;
    multiRegionProps2.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> clusters;
    clusters.push_back(cluster1.GetArn());
    multiRegionProps2.SetClusters(clusters);

    tags["Name"] = "cpp multi region cluster 2";
    createClusterRequest2.SetMultiRegionProperties(multiRegionProps2);
    createClusterRequest2.SetTags(tags);
    createClusterRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());
```

```
    auto createOutcome2 = client2.CreateCluster(createClusterRequest2);
    if (!createOutcome2.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region2 << ": "
                  << createOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster2 = createOutcome2.GetResult();
    std::cout << "Created " << cluster2.GetArn() << std::endl;

    // Now that we know the cluster2 arn we can set it as a peer of cluster1
    UpdateClusterRequest updateClusterRequest;
    updateClusterRequest.SetIdentifier(cluster1.GetIdentifier());

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> updatedClusters;
    updatedClusters.push_back(cluster2.GetArn());
    updatedProps.SetClusters(updatedClusters);

    updateClusterRequest.SetMultiRegionProperties(updatedProps);
    updateClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto updateOutcome = client1.UpdateCluster(updateClusterRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster in " << region1 << ": "
                  << updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to update multi-region clusters");
    }

    std::cout << "Added " << cluster2.GetArn() << " as a peer of " <<
 cluster1.GetArn() << std::endl;

    return std::make_pair(cluster1, cluster2);
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define regions for the multi-region setup
            Aws::String region1 = "us-east-1";
```

```
            Aws::String region2 = "us-east-2";
            Aws::String witnessRegion = "us-west-2";

            auto [cluster1, cluster2] = CreateMultiRegionClusters(region1, region2,
 witnessRegion);

            std::cout << "Created multi region clusters:" << std::endl;
            std::cout << "Cluster 1 ARN: " << cluster1.GetArn() << std::endl;
            std::cout << "Cluster 2 ARN: " << cluster2.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
import { DSQLClient, CreateClusterCommand, UpdateClusterCommand,
 waitUntilClusterActive } from "@aws-sdk/client-dsql";

async function createMultiRegionCluster(region1, region2, witnessRegion) {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        // We can only set the witness region for the first cluster
        console.log(`Creating cluster in ${region1}`);
        const createClusterCommand1 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 1"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion
            }
        });
```

```
        const response1 = await client1.send(createClusterCommand1);
        console.log(`Created ${response1.arn}`);

        // For the second cluster we can set witness region and designate the first
cluster as a peer
        console.log(`Creating cluster in ${region2}`);
        const createClusterCommand2 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 2"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response1.arn]
            }
        });
        const response2 = await client2.send(createClusterCommand2);
        console.log(`Created ${response2.arn}`);

        // Now that we know the second cluster arn we can set it as a peer of the
first cluster
        const updateClusterCommand = new UpdateClusterCommand({
            identifier: response1.identifier,
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response2.arn]
            }
        });
        await client1.send(updateClusterCommand);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters they'll
begin the transition to ACTIVE
        console.log(`Waiting for cluster ${response1.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);
```

```
        console.log(`Waiting for cluster ${response2.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const region2 = "us-east-2";
    const witnessRegion = "us-west-2";

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
```

```
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;

import java.time.Duration;
import java.util.Map;

public class CreateMultiRegionCluster {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        Region region2 = Region.US_EAST_2;
        Region witnessRegion = Region.US_WEST_2;

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            // We can only set the witness region for the first cluster
            System.out.println("Creating cluster in " + region1);
            CreateClusterRequest request1 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->
 mrp.witnessRegion(witnessRegion.toString()))
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster1 = client1.createCluster(request1);
            System.out.println("Created " + cluster1.arn());

            // For the second cluster we can set the witness region and designate
            // cluster1 as a peer.
            System.out.println("Creating cluster in " + region2);
            CreateClusterRequest request2 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->

 mrp.witnessRegion(witnessRegion.toString()).clusters(cluster1.arn())
                    )
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster2 = client2.createCluster(request2);
```

```
            System.out.println("Created " + cluster2.arn());

            // Now that we know the cluster2 ARN we can set it as a peer of cluster1
            UpdateClusterRequest updateReq = UpdateClusterRequest.builder()
                    .identifier(cluster1.identifier())
                    .multiRegionProperties(mrp ->

mrp.witnessRegion(witnessRegion.toString()).clusters(cluster2.arn())
                    )
                    .build();
            client1.updateCluster(updateReq);
            System.out.printf("Added %s as a peer of %s%n", cluster2.arn(),
cluster1.arn());

            // Now that MultiRegionProperties is fully defined for both clusters
they'll begin
            // the transition to ACTIVE.
            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
cluster1.arn());
            GetClusterResponse activeCluster1 =
client1.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster1.identifier()),
                    config -> config.backoffStrategyV2(

BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
cluster2.arn());
            GetClusterResponse activeCluster2 =
client2.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster2.identifier()),
                    config -> config.backoffStrategyV2(

BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.println("Created multi region clusters:");
            System.out.println(activeCluster1);
            System.out.println(activeCluster2);
        }
    }
```

```
}
```

Rust

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::types::MultiRegionProperties;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_multi_region_clusters(
    region_1: &'static str,
    region_2: &'static str,
    witness_region: &'static str,
) -> (GetClusterOutput, GetClusterOutput) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;
```

```
    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust multi region cluster"),
    )]);

    // We can only set the witness region for the first cluster
    println!("Creating cluster in {region_1}");
    let cluster_1 = client_1
        .create_cluster()
        .set_tags(Some(tags.clone()))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_1_arn = &cluster_1.arn;
    println!("Created {cluster_1_arn}");

    // For the second cluster we can set witness region and designate cluster_1 as a
peer
    println!("Creating cluster in {region_2}");
    let cluster_2 = client_2
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_1.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_2_arn = &cluster_2.arn;
    println!("Created {cluster_2_arn}");

    // Now that we know the cluster_2 arn we can set it as a peer of cluster_1
    client_1
        .update_cluster()
```

```
        .identifier(&cluster_1.identifier)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_2.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    println!("Added {cluster_2_arn} as a peer of {cluster_1_arn}");

    // Now that the multi-region properties are fully defined for both clusters
    // they'll begin the transition to ACTIVE
    println!("Waiting for {cluster_1_arn} to become ACTIVE");
    let cluster_1_output = client_1
        .wait_until_cluster_active()
        .identifier(&cluster_1.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    println!("Waiting for {cluster_2_arn} to become ACTIVE");
    let cluster_2_output = client_2
        .wait_until_cluster_active()
        .identifier(&cluster_2.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    (cluster_1_output, cluster_2_output)
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let region_2 = "us-east-2";
    let witness_region = "us-west-2";

    let (cluster_1, cluster_2) =
```

```
        create_multi_region_clusters(region_1, region_2, witness_region).await;

    println!("Created multi region clusters:");
    println!("{:#?}", cluster_1);
    println!("{:#?}", cluster_2);

    Ok(())
}
```

Ruby

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
require "aws-sdk-dsql"
require "pp"

def create_multi_region_clusters(region_1, region_2, witness_region)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  # We can only set the witness region for the first cluster
  puts "Creating cluster in #{region_1}"
  cluster_1 = client_1.create_cluster(
    deletion_protection_enabled: true,
    multi_region_properties: {
      witness_region: witness_region
    },
    tags: {
      Name: "ruby multi region cluster"
    }
  )
  puts "Created #{cluster_1.arn}"

  # For the second cluster we can set witness region and designate cluster_1 as a
 peer
  puts "Creating cluster in #{region_2}"
  cluster_2 = client_2.create_cluster(
    deletion_protection_enabled: true,
    multi_region_properties: {
      witness_region: witness_region,
      clusters: [ cluster_1.arn ]
```

```
    },
    tags: {
      Name: "ruby multi region cluster"
    }
  )
  puts "Created #{cluster_2.arn}"

  # Now that we know the cluster_2 arn we can set it as a peer of cluster_1
  client_1.update_cluster(
    identifier: cluster_1.identifier,
    multi_region_properties: {
      witness_region: witness_region,
      clusters: [ cluster_2.arn ]
    }
  )
  puts "Added #{cluster_2.arn} as a peer of #{cluster_1.arn}"

  # Now that multi_region_properties is fully defined for both clusters
  # they'll begin the transition to ACTIVE
  puts "Waiting for #{cluster_1.arn} to become ACTIVE"
  cluster_1 = client_1.wait_until(:cluster_active, identifier: cluster_1.identifier)
 do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end

  puts "Waiting for #{cluster_2.arn} to become ACTIVE"
  cluster_2 = client_2.wait_until(:cluster_active, identifier: cluster_2.identifier)
 do |w|
    w.max_attempts = 30
    w.delay = 10
  end

  [ cluster_1, cluster_2 ]
rescue Aws::Errors::ServiceError => e
  abort "Failed to create multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  region_2 = "us-east-2"
  witness_region = "us-west-2"
```

```
  cluster_1, cluster_2 = create_multi_region_clusters(region_1, region_2,
 witness_region)

  puts "Created multi region clusters:"
  pp cluster_1
  pp cluster_2
end

main if $PROGRAM_NAME == __FILE__
```

Golang

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
 dtypes "github.com/aws/aws-sdk-go-v2/service/dsql/types"
)

func CreateMultiRegionClusters(ctx context.Context, witness, region1, region2
 string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Create a DSQL region 1 client
 client := dsql.NewFromConfig(cfg)

 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
 if err != nil {
```

```
 log.Fatalf("Failed to load AWS configuration: %v", err)
}

// Create a DSQL region 2 client
client2 := dsql.NewFromConfig(cfg2, func(o *dsql.Options) {
 o.Region = region2
})

// Create cluster
deleteProtect := true

// We can only set the witness region for the first cluster
input := &dsql.CreateClusterInput{
 DeletionProtectionEnabled: &deleteProtect,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String(witness),
 },
 Tags: map[string]string{
  "Name": "go multi-region cluster",
 },
}

clusterProperties, err := client.CreateCluster(context.Background(), input)

if err != nil {
 return fmt.Errorf("failed to create first cluster: %v", err)
}

// create second cluster
cluster2Arns := []string{*clusterProperties.Arn}

// For the second cluster we can set witness region and designate the first cluster
as a peer
input2 := &dsql.CreateClusterInput{
 DeletionProtectionEnabled: &deleteProtect,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster2Arns,
 },
 Tags: map[string]string{
  "Name": "go multi-region cluster",
 },
}
```

```
clusterProperties2, err := client2.CreateCluster(context.Background(), input2)

if err != nil {
 return fmt.Errorf("failed to create second cluster: %v", err)
}

// link initial cluster to second cluster
cluster1Arns := []string{*clusterProperties2.Arn}

// Now that we know the second cluster arn we can set it as a peer of the first
cluster
input3 := dsql.UpdateClusterInput{
 Identifier: clusterProperties.Identifier,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster1Arns,
 }}

_, err = client.UpdateCluster(context.Background(), &input3)

if err != nil {
 return fmt.Errorf("failed to update cluster to associate with first cluster. %v",
err)
}

// Create the waiter with our custom options for first cluster
waiter := dsql.NewClusterActiveWaiter(client, func(o
*dsql.ClusterActiveWaiterOptions) {
 o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
minutes
 o.MinDelay = 10 * time.Second
 o.LogWaitAttempts = true
})

// Now that multiRegionProperties is fully defined for both clusters
// they'll begin the transition to ACTIVE

// Create the input for the clusterProperties to monitor for first cluster
getInput := &dsql.GetClusterInput{
 Identifier: clusterProperties.Identifier,
}

// Wait for the first cluster to become active
```

```
 fmt.Printf("Waiting for first cluster %s to become active...\n",
 *clusterProperties.Identifier)
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for first cluster to become active: %w", err)
 }

 // Create the waiter with our custom options
 waiter2 := dsql.NewClusterActiveWaiter(client2, func(o
 *dsql.ClusterActiveWaiterOptions) {
  o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
 minutes
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })

 // Create the input for the clusterProperties to monitor for second
 getInput2 := &dsql.GetClusterInput{
  Identifier: clusterProperties2.Identifier,
 }

 // Wait for the second cluster to become active
 fmt.Printf("Waiting for second cluster %s to become active...\n",
 *clusterProperties2.Identifier)
 err = waiter2.Wait(ctx, getInput2, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for second cluster to become active: %w", err)
 }

 fmt.Printf("Cluster %s is now active\n", *clusterProperties.Identifier)
 fmt.Printf("Cluster %s is now active\n", *clusterProperties2.Identifier)
 return nil
}

// Example usage in main function
func main() {
 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := CreateMultiRegionClusters(ctx, "us-west-2", "us-east-1", "us-east-2")
 if err != nil {
  fmt.Printf("failed to create multi-region clusters: %v", err)
  panic(err)
```

```
 }

}
```

.NET

Per creare un cluster multi-Regione, attenersi all'esempio descritto di seguito. La creazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class CreateMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create multi-region clusters with a witness region.
        /// </summary>
```

```
        public static async Task<(CreateClusterResponse, CreateClusterResponse)>
Create(
            RegionEndpoint region1,
            RegionEndpoint region2,
            RegionEndpoint witnessRegion)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp multi region cluster" }
                };

                // We can only set the witness region for the first cluster
                var createClusterRequest1 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName
                    }
                };

                var cluster1 = await
client1.CreateClusterAsync(createClusterRequest1);
                var cluster1Arn = cluster1.Arn;
                Console.WriteLine($"Initiated creation of {cluster1Arn}");

                // For the second cluster we can set witness region and designate
cluster1 as a peer
                var createClusterRequest2 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster1.Arn }
                    }
                };
```

```
                var cluster2 = await
 client2.CreateClusterAsync(createClusterRequest2);
                var cluster2Arn = cluster2.Arn;
                Console.WriteLine($"Initiated creation of {cluster2Arn}");

                // Now that we know the cluster2 arn we can set it as a peer of
 cluster1
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = cluster1.Identifier,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster2.Arn }
                    }
                };

                await client1.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Added {cluster2Arn} as a peer of
 {cluster1Arn}");

                return (cluster1, cluster2);
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var region2 = RegionEndpoint.USEast2;
            var witnessRegion = RegionEndpoint.USWest2;

            var (cluster1, cluster2) = await Create(region1, region2,
 witnessRegion);

            Console.WriteLine("Created multi region clusters:");
            Console.WriteLine($"Cluster 1: {cluster1.Arn}");
            Console.WriteLine($"Cluster 2: {cluster2.Arn}");
        }
    }
}
```

Recupero delle informazioni di un cluster

Negli esempi seguenti viene illustrato come recuperare le informazioni su un cluster multi-Regione utilizzando diversi linguaggi di programmazione.

Python

Per ottenere informazioni su un cluster multi-Regione, attenersi all'esempio descritto di seguito.

```
import boto3
from datetime import datetime
import json


def get_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.get_cluster(identifier=identifier)
    except:
        print(f"Unable to get cluster {identifier} in region {region}")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    response = get_cluster(region, cluster_id)

    print(json.dumps(response, indent=2, default=lambda obj: obj.isoformat() if
 isinstance(obj, datetime) else None))


if __name__ == "__main__":
    main()
```

C++

Utilizzare l'esempio seguente per ottenere informazioni su un cluster multi-Regione.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
```

```
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Retrieves information about a cluster in Amazon Aurora DSQL
 */
GetClusterResult GetCluster(const Aws::String& region, const Aws::String&
 identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Get the cluster
    GetClusterRequest getClusterRequest;
    getClusterRequest.SetIdentifier(identifier);

    auto getOutcome = client.GetCluster(getClusterRequest);
    if (!getOutcome.IsSuccess()) {
        std::cerr << "Failed to retrieve cluster " << identifier << " in " << region
 << ": "
                  << getOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to retrieve cluster " + identifier + " in
 region " + region);
    }

    return getOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            auto cluster = GetCluster(region, clusterId);
```

```
            // Print cluster details
            std::cout << "Cluster Details:" << std::endl;
            std::cout << "ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per ottenere informazioni su un cluster multi-Regione, attenersi all'esempio descritto di seguito.

```
import { DSQLClient, GetClusterCommand } from "@aws-sdk/client-dsql";

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  const getClusterCommand = new GetClusterCommand({
    identifier: clusterId,
  });

  try {
    return await client.send(getClusterCommand);
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or deleted");
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  const response = await getCluster(region, clusterId);
```

```
  console.log("Cluster: ", response);
}

main();
```

Java

L'esempio seguente consente di ottenere informazioni su un cluster multi-Regione.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

public class GetCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            GetClusterResponse cluster = client.getCluster(r ->
 r.identifier(clusterId));
            System.out.println(cluster);
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

L'esempio seguente consente di ottenere informazioni su un cluster multi-Regione.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Get a ClusterResource from DSQL cluster identifier
pub async fn get_cluster(region: &'static str, identifier: &'static str) ->
 GetClusterOutput {
    let client = dsql_client(region).await;
    client
        .get_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = get_cluster(region, "<your cluster id>").await;
```

```
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

L'esempio seguente consente di ottenere informazioni su un cluster multi-Regione.

```
require "aws-sdk-dsql"
require "pp"

def get_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  client.get_cluster(identifier: identifier)
rescue Aws::Errors::ServiceError => e
  abort "Unable to retrieve cluster #{identifier} in region #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  cluster = get_cluster(region, cluster_id)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

L'esempio seguente consente di ottenere informazioni su un cluster multi-Regione.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
```

```
   public class GetCluster
   {
       /// <summary>
       /// Create a client. We will use this later for performing operations on the
cluster.
       /// </summary>
       private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
       {
           var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
           var clientConfig = new AmazonDSQLConfig
           {
               RegionEndpoint = region
           };
           return new AmazonDSQLClient(awsCredentials, clientConfig);
       }

       /// <summary>
       /// Get information about a DSQL cluster.
       /// </summary>
       public static async Task<GetClusterResponse> Get(RegionEndpoint region,
string identifier)
       {
           using (var client = await CreateDSQLClient(region))
           {
               var getClusterRequest = new GetClusterRequest
               {
                   Identifier = identifier
               };

               return await client.GetClusterAsync(getClusterRequest);
           }
       }

       private static async Task Main()
       {
           var region = RegionEndpoint.USEast1;
           var clusterId = "<your cluster id>";

           var response = await Get(region, clusterId);
           Console.WriteLine($"Cluster ARN: {response.Arn}");
       }
   }
```

```
}
```

Golang

L'esempio seguente consente di ottenere informazioni su un cluster multi-Regione.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func GetCluster(ctx context.Context, region, identifier string) (clusterStatus
 *dsql.GetClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := &dsql.GetClusterInput{
  Identifier: aws.String(identifier),
 }
 clusterStatus, err = client.GetCluster(context.Background(), input)

 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }

 log.Printf("Cluster ARN: %s", *clusterStatus.Arn)

 return clusterStatus, nil
}
```

```
func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 _, err := GetCluster(ctx, region, identifier)
 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }
}
```

Aggiornamento di un cluster

Negli esempi seguenti viene illustrato come aggiornare un cluster multi-Regione utilizzando diversi linguaggi di programmazione.

Python

Per aggiornare un cluster multi-Regione, attenersi all'esempio descritto di seguito.

```
import boto3


def update_cluster(region, cluster_id, deletion_protection_enabled):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.update_cluster(identifier=cluster_id,
 deletionProtectionEnabled=deletion_protection_enabled)
    except:
        print("Unable to update cluster")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    deletion_protection_enabled = False
    response = update_cluster(region, cluster_id, deletion_protection_enabled)
```

```
    print(f"Updated {response["arn"]} with deletion_protection_enabled:
 {deletion_protection_enabled}")


if __name__ == "__main__":
    main()
```

C++

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Updates a cluster in Amazon Aurora DSQL
 */
UpdateClusterResult UpdateCluster(const Aws::String& region, const
 Aws::Map<Aws::String, Aws::String>& updateParams) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create update request
    UpdateClusterRequest updateRequest;
    updateRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Set identifier (required)
    if (updateParams.find("identifier") != updateParams.end()) {
        updateRequest.SetIdentifier(updateParams.at("identifier"));
    } else {
        throw std::runtime_error("Cluster identifier is required for update
 operation");
    }
```

```
    // Set deletion protection if specified
    if (updateParams.find("deletion_protection_enabled") != updateParams.end()) {
        bool deletionProtection = (updateParams.at("deletion_protection_enabled") ==
 "true");
        updateRequest.SetDeletionProtectionEnabled(deletionProtection);
    }

    // Execute the update
    auto updateOutcome = client.UpdateCluster(updateRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster: " <<
 updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to update cluster");
    }

    return updateOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and update parameters
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            // Create parameter map
            Aws::Map<Aws::String, Aws::String> updateParams;
            updateParams["identifier"] = clusterId;
            updateParams["deletion_protection_enabled"] = "false";

            auto updatedCluster = UpdateCluster(region, updateParams);

            std::cout << "Updated " << updatedCluster.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per aggiornare un cluster multi-Regione, attenersi all'esempio descritto di seguito.

```
import { DSQLClient, UpdateClusterCommand } from "@aws-sdk/client-dsql";

export async function updateCluster(region, clusterId, deletionProtectionEnabled) {

  const client = new DSQLClient({ region });

  const updateClusterCommand = new UpdateClusterCommand({
    identifier: clusterId,
    deletionProtectionEnabled: deletionProtectionEnabled
  });

  try {
    return await client.send(updateClusterCommand);
  } catch (error) {
    console.error("Unable to update cluster", error.message);
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";
  const deletionProtectionEnabled = false;

  const response = await updateCluster(region, clusterId,
 deletionProtectionEnabled);
  console.log(`Updated ${response.arn}`);
}

main();
```

Java

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
package org.example;
```

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;
import software.amazon.awssdk.services.dsql.model.UpdateClusterResponse;

public class UpdateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            UpdateClusterRequest request = UpdateClusterRequest.builder()
                    .identifier(clusterId)
                    .deletionProtectionEnabled(false)
                    .build();
            UpdateClusterResponse cluster = client.updateCluster(request);
            System.out.println("Updated " + cluster.arn());
        }
    }
}
```

Rust

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::update_cluster::UpdateClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;
```

```
    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Update a DSQL cluster and set delete protection to false. Also add new tags.
pub async fn update_cluster(region: &'static str, identifier: &'static str) ->
 UpdateClusterOutput {
    let client = dsql_client(region).await;
    // Update delete protection
    let update_response = client
        .update_cluster()
        .identifier(identifier)
        .deletion_protection_enabled(false)
        .send()
        .await
        .unwrap();

    update_response
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = update_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
require "aws-sdk-dsql"

def update_cluster(region, update_params)
  client = Aws::DSQL::Client.new(region: region)
  client.update_cluster(update_params)
rescue Aws::Errors::ServiceError => e
  abort "Unable to update cluster: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  updated_cluster = update_cluster(region, {
    identifier: cluster_id,
    deletion_protection_enabled: false
  })
  puts "Updated #{updated_cluster.arn}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class UpdateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
```

```
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Update a DSQL cluster and set delete protection to false.
        /// </summary>
        public static async Task<UpdateClusterResponse> Update(RegionEndpoint
 region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = identifier,
                    DeletionProtectionEnabled = false
                };

                UpdateClusterResponse response = await
 client.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Updated {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            await Update(region, clusterId);
        }
    }
}
```

Golang

Utilizzare l'esempio seguente per aggiornare un cluster multi-Regione.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func UpdateCluster(ctx context.Context, region, id string, deleteProtection bool)
 (clusterStatus *dsql.UpdateClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := dsql.UpdateClusterInput{
  Identifier:                &id,
  DeletionProtectionEnabled: &deleteProtection,
 }

 clusterStatus, err = client.UpdateCluster(context.Background(), &input)

 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }

 log.Printf("Cluster updated successfully: %v", clusterStatus.Status)
 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()
```

```
 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"
 deleteProtection := false

 _, err := UpdateCluster(ctx, region, identifier, deleteProtection)
 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }
}
```

Eliminazione di un cluster

Negli esempi seguenti viene illustrato come eliminare un cluster multi-Regione utilizzando diversi linguaggi di programmazione.

Python

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
import boto3


def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2):
    try:

        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        client_1.delete_cluster(identifier=cluster_id_1)
        print(f"Deleting cluster {cluster_id_1} in {region_1}")

        # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted

        client_2.delete_cluster(identifier=cluster_id_2)
        print(f"Deleting cluster {cluster_id_2} in {region_2}")

        # Now that both clusters have been marked for deletion they will transition
        # to DELETING state and finalize deletion
        print(f"Waiting for {cluster_id_1} to finish deletion")
```

```
        client_1.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_1,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_id_2} to finish deletion")
        client_2.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_2,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

    except:
        print("Unable to delete cluster")
        raise


def main():
    region_1 = "us-east-1"
    cluster_id_1 = "<cluster 1 id>"
    region_2 = "us-east-2"
    cluster_id_2 = "<cluster 2 id>"

    delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
    print(f"Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in {region_2}")


if __name__ == "__main__":
    main()
```

C++

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
```

```
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes multi-region clusters in Amazon Aurora DSQL
 */
void DeleteMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& clusterId1,
    const Aws::String& region2,
    const Aws::String& clusterId2) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    // Delete the first cluster
    std::cout << "Deleting cluster " << clusterId1 << " in " << region1 <<
 std::endl;

    DeleteClusterRequest deleteRequest1;
    deleteRequest1.SetIdentifier(clusterId1);
    deleteRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome1 = client1.DeleteCluster(deleteRequest1);
    if (!deleteOutcome1.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId1 << " in " << region1
 << ": "
                  << deleteOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }
```

```
    // cluster1 will stay in PENDING_DELETE state until cluster2 is deleted
    std::cout << "Deleting cluster " << clusterId2 << " in " << region2 <<
 std::endl;

    DeleteClusterRequest deleteRequest2;
    deleteRequest2.SetIdentifier(clusterId2);
    deleteRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome2 = client2.DeleteCluster(deleteRequest2);
    if (!deleteOutcome2.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId2 << " in " << region2
 << ": "
                  << deleteOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            Aws::String region1 = "us-east-1";
            Aws::String clusterId1 = "<your cluster id 1>";
            Aws::String region2 = "us-east-2";
            Aws::String clusterId2 = "<your cluster id 2>";

            DeleteMultiRegionClusters(region1, clusterId1, region2, clusterId2);

            std::cout << "Deleted " << clusterId1 << " in " << region1
                      << " and " << clusterId2 << " in " << region2 << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteMultiRegionClusters(region1, cluster1_id, region2, cluster2_id)
 {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        const deleteClusterCommand1 = new DeleteClusterCommand({
            identifier: cluster1_id,
        });
        const response1 = await client1.send(deleteClusterCommand1);

        const deleteClusterCommand2 = new DeleteClusterCommand({
            identifier: cluster2_id,
        });
        const response2 = await client2.send(deleteClusterCommand2);

        console.log(`Waiting for cluster1 ${response1.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster1 Id ${response1.identifier} is now deleted`);

        console.log(`Waiting for cluster2 ${response2.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
```

```
        );
        console.log(`Cluster2 Id ${response2.identifier} is now deleted`);
        return;
    } catch (error) {
        if (error.name === "ResourceNotFoundException") {
            console.log("Some or all Cluster ARNs not found or already deleted");
        } else {
            console.error("Unable to delete multi-region clusters: ",
 error.message);
        }
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const cluster1_id = "<CLUSTER_ID_1>";
    const region2 = "us-east-2";
    const cluster2_id = "<CLUSTER_ID_2>";

    const response = await deleteMultiRegionClusters(region1, cluster1_id, region2,
 cluster2_id);
    console.log(`Deleted ${cluster1_id} in ${region1} and ${cluster2_id} in
 ${region2}`);
}

main();
```

Java

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.DeleteClusterRequest;

import java.time.Duration;
```

```
public class DeleteMultiRegionClusters {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        String clusterId1 = "<your cluster id 1>";
        Region region2 = Region.US_EAST_2;
        String clusterId2 = "<your cluster id 2>";

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            System.out.printf("Deleting cluster %s in %s%n", clusterId1, region1);
            DeleteClusterRequest request1 = DeleteClusterRequest.builder()
                    .identifier(clusterId1)
                    .build();
            client1.deleteCluster(request1);

            // cluster1 will stay in PENDING_DELETE until cluster2 is deleted
            System.out.printf("Deleting cluster %s in %s%n", clusterId2, region2);
            DeleteClusterRequest request2 = DeleteClusterRequest.builder()
                    .identifier(clusterId2)
                    .build();
            client2.deleteCluster(request2);

            // Now that both clusters have been marked for deletion they will
 transition
            // to DELETING state and finalize deletion.
            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId1);
            client1.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId1),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );

            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId2);
```

```
            client2.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId2),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );

            System.out.printf("Deleted %s in %s and %s in %s%n", clusterId1,
 region1, clusterId2, region2);
        }
    }
}
```

Rust

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{Client, Config};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
```

```
pub async fn delete_multi_region_clusters(
    region_1: &'static str,
    cluster_id_1: &'static str,
    region_2: &'static str,
    cluster_id_2: &'static str,
) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;

    println!("Deleting cluster {cluster_id_1} in {region_1}");
    client_1
        .delete_cluster()
        .identifier(cluster_id_1)
        .send()
        .await
        .unwrap();

    // cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
    println!("Deleting cluster {cluster_id_2} in {region_2}");
    client_2
        .delete_cluster()
        .identifier(cluster_id_2)
        .send()
        .await
        .unwrap();

    // Now that both clusters have been marked for deletion they will transition
    // to DELETING state and finalize deletion
    println!("Waiting for {cluster_id_1} to finish deletion");
    client_1
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_1)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();

    println!("Waiting for {cluster_id_2} to finish deletion");
    client_2
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_2)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}
```

```
#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let cluster_id_1 = "<cluster 1 to be deleted>";
    let region_2 = "us-east-2";
    let cluster_id_2 = "<cluster 2 to be deleted>";

    delete_multi_region_clusters(region_1, cluster_id_1, region_2,
 cluster_id_2).await;
    println!("Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in
 {region_2}");

    Ok(())
}
```

Ruby

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
require "aws-sdk-dsql"

def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  puts "Deleting cluster #{cluster_id_1} in #{region_1}"
  client_1.delete_cluster(identifier: cluster_id_1)

  # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
  puts "Deleting #{cluster_id_2} in #{region_2}"
  client_2.delete_cluster(identifier: cluster_id_2)

  # Now that both clusters have been marked for deletion they will transition
  # to DELETING state and finalize deletion
  puts "Waiting for #{cluster_id_1} to finish deletion"
  client_1.wait_until(:cluster_not_exists, identifier: cluster_id_1) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
  puts "Waiting for #{cluster_id_2} to finish deletion"
  client_2.wait_until(:cluster_not_exists, identifier: cluster_id_2) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Failed to delete multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  cluster_id_1 = "<your cluster id 1>"
  region_2 = "us-east-2"
  cluster_id_2 = "<your cluster id 2>"

  delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  puts "Deleted #{cluster_id_1} in #{region_1} and #{cluster_id_2} in #{region_2}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class DeleteMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
```

```
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
        {
            var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete multi-region clusters.
        /// </summary>
        public static async Task Delete(
            RegionEndpoint region1,
            string clusterId1,
            RegionEndpoint region2,
            string clusterId2)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var deleteRequest1 = new DeleteClusterRequest
                {
                    Identifier = clusterId1
                };

                var deleteResponse1 = await
client1.DeleteClusterAsync(deleteRequest1);
                Console.WriteLine($"Initiated deletion of {deleteResponse1.Arn}");

                // cluster 1 will stay in PENDING_DELETE state until cluster 2 is
deleted
                var deleteRequest2 = new DeleteClusterRequest
                {
                    Identifier = clusterId2
                };

                var deleteResponse2 = await
client2.DeleteClusterAsync(deleteRequest2);
                Console.WriteLine($"Initiated deletion of {deleteResponse2.Arn}");
```

```
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var cluster1 = "<cluster 1 to be deleted>";
            var region2 = RegionEndpoint.USEast2;
            var cluster2 = "<cluster 2 to be deleted>";

            await Delete(region1, cluster1, region2, cluster2);
        }
    }
}
```

Golang

Per eliminare un cluster multi-Regione, attenersi all'esempio descritto di seguito. L'eliminazione di un cluster multi-Regione potrebbe richiedere alcuni minuti.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func DeleteMultiRegionClusters(ctx context.Context, region1, clusterId1, region2,
 clusterId2 string) error {
 // Load the AWS configuration for region 1
 cfg1, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  return fmt.Errorf("unable to load SDK config for region %s: %w", region1, err)
 }

 // Load the AWS configuration for region 2
 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
```

```
if err != nil {
 return fmt.Errorf("unable to load SDK config for region %s: %w", region2, err)
}

// Create DSQL clients for both regions
client1 := dsql.NewFromConfig(cfg1)
client2 := dsql.NewFromConfig(cfg2)

// Delete cluster in region 1
fmt.Printf("Deleting cluster %s in %s\n", clusterId1, region1)
_, err = client1.DeleteCluster(ctx, &dsql.DeleteClusterInput{
 Identifier: aws.String(clusterId1),
})
if err != nil {
 return fmt.Errorf("failed to delete cluster in region %s: %w", region1, err)
}

// Delete cluster in region 2
fmt.Printf("Deleting cluster %s in %s\n", clusterId2, region2)
_, err = client2.DeleteCluster(ctx, &dsql.DeleteClusterInput{
 Identifier: aws.String(clusterId2),
})
if err != nil {
 return fmt.Errorf("failed to delete cluster in region %s: %w", region2, err)
}

// Create waiters for both regions
waiter1 := dsql.NewClusterNotExistsWaiter(client1, func(options
*dsql.ClusterNotExistsWaiterOptions) {
 options.MinDelay = 10 * time.Second
 options.MaxDelay = 30 * time.Second
 options.LogWaitAttempts = true
})

waiter2 := dsql.NewClusterNotExistsWaiter(client2, func(options
*dsql.ClusterNotExistsWaiterOptions) {
 options.MinDelay = 10 * time.Second
 options.MaxDelay = 30 * time.Second
 options.LogWaitAttempts = true
})

// Wait for cluster in region 1 to be deleted
fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId1)
err = waiter1.Wait(ctx, &dsql.GetClusterInput{
```

```
  Identifier: aws.String(clusterId1),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region1,
 err)
 }

 // Wait for cluster in region 2 to be deleted
 fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId2)
 err = waiter2.Wait(ctx, &dsql.GetClusterInput{
  Identifier: aws.String(clusterId2),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region2,
 err)
 }

 fmt.Printf("Successfully deleted clusters %s in %s and %s in %s\n",
  clusterId1, region1, clusterId2, region2)
 return nil
}

// Example usage in main function
func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := DeleteMultiRegionClusters(
  ctx,
  "us-east-1",      // region1
  "<CLUSTER_ID_1>", // clusterId1
  "us-east-2",      // region2
  "<CLUSTER_ID_2>", // clusterId2
 )
 if err != nil {
  log.Fatalf("Failed to delete multi-region clusters: %v", err)
 }
}
```

Per altri esempi di codice, visitare il repository GitHub degli esempi di Aurora DSQL.

Utilizzo della CLI di AWS

La CLI di AWS fornisce un'interfaccia a riga di comando per la gestione dei cluster Aurora DSQL multi-Regione. Negli esempi seguenti viene illustrato come creare, configurare ed eliminare cluster multi-Regione.

Connessione a un cluster multi-Regione

I cluster in peering multi-Regione forniscono due endpoint regionali, uno per ogni cluster Regione AWS in peering. Entrambi gli endpoint presentano un unico database logico che supporta operazioni di lettura e scrittura simultanee con una forte coerenza dei dati. Oltre ai cluster in peering, un cluster multi-Regione dispone anche di una Regione di riferimento che archivia una finestra limitata di registri delle transazioni crittografati, utilizzata per migliorare la durabilità e la disponibilità multi-Regione. Le regioni testimone dei cluster multi-Regione non dispongono di endpoint.

Creazione di cluster multi-Regione

Per creare cluster multi-Regione, è prima necessario creare un cluster con una Regione testimone. Quindi si esegue il peering di questo cluster a un secondo cluster che condivide la stessa Regione testimone del primo cluster. Nell'esempio seguente viene illustrato come creare cluster negli Stati Uniti orientali (Virginia settentrionale) e negli Stati Uniti orientali (Ohio) con Stati Uniti occidentali (Oregon) come Regione testimone.

Fase 1: creazione di un cluster negli Stati Uniti orientali (Virginia settentrionale)

Per creare un cluster nella Regione AWS Stati Uniti orientali (Virginia settentrionale) con proprietà multi-Regione, utilizzare il seguente comando.

```
aws dsql create-cluster \
--region us-east-1 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Risposta:

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
```

```
        "encryptionStatus": "ENABLED"
    }
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

> **Note**
>
> Quando l'operazione API ha esito positivo, il cluster entra nello stato `PENDING_SETUP`. La creazione del cluster rimane in stato `PENDING_SETUP` finché non si aggiorna il cluster con l'ARN del relativo cluster peer.

Fase 2: creazione di un secondo cluster negli Stati Uniti orientali (Ohio)

Per creare un cluster nella Regione AWS Stati Uniti orientali (Ohio) con proprietà multi-Regione, utilizzare il comando seguente.

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Risposta:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_SETUP",
    "creationTime": "2025-05-06T06:51:16.145000-07:00",
    "deletionProtectionEnabled": true,
    "multiRegionProperties": {
        "witnessRegion": "us-west-2",
        "clusters": [
            "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
        ]
    }
}
```

Quando l'operazione API ha esito positivo, il cluster passa allo stato `PENDING_SETUP`. La creazione del cluster rimane nello stato `PENDING_SETUP` fino a quando non viene aggiornata con l'ARN di un altro cluster in peering.

Fase 3: peering dei cluster negli Stati Uniti orientali (Virginia settentrionale) con quello negli Stati Uniti orientali (Ohio)

Per effettuare il peering del cluster negli Stati Uniti orientali (Virginia settentrionale) con il cluster negli Stati Uniti orientali (Ohio), utilizzare il comando `update-cluster`. Specificate il nome del cluster negli Stati Uniti orientali (Virginia settentrionale) e una stringa JSON con l'ARN del cluster negli Stati Uniti orientali (Ohio).

```
aws dsql update-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4' \
--multi-region-properties '{"witnessRegion": "us-west-2","clusters": ["arn:aws:dsql:us-
east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"]}'
```

Example Risposta

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

Fase 4: peering del cluster negli Stati Uniti orientali (Ohio) con quello negli Stati Uniti orientali (Virginia settentrionale)

Per effettuare il peering tra il cluster negli Stati Uniti orientali (Ohio) e il cluster negli Stati Uniti orientali (Virginia settentrionale), utilizza il comando `update-cluster`. Specificare il nome del cluster negli Stati Uniti orientali (Ohio) e una stringa JSON con l'ARN del cluster negli Stati Uniti orientali (Virginia settentrionale).

Example

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5' \
--multi-region-properties '{"witnessRegion": "us-west-2", "clusters":
 ["arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Example Risposta

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:51:16.145000-07:00"
}
```

> **Note**
>
> Se il peering si conclude con successo, entrambi i cluster passano dallo stato "IN ATTESA DI CONFIGURAZIONE" a "CREAZIONE IN CORSO" e infine allo stato "ATTIVO" quando sono pronti per l'uso.

Visualizzazione delle proprietà del cluster multi-Regione

Quando si descrive un cluster, è possibile visualizzare le proprietà multi-Regione per i cluster in diverse Regioni AWS.

Example

```
aws dsql get-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4'
```

Example Risposta

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_SETUP",
    "encryptionDetails": {
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "encryptionStatus": "ENABLED"
},
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "multiRegionProperties": {
```

```
        "witnessRegion": "us-west-2",
        "clusters": [
           "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
           "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
        ]
    }
}
```

Peering dei cluster durante la creazione

È possibile ridurre il numero di passaggi includendo le informazioni di peering durante la creazione del cluster. Dopo aver creato il primo cluster negli Stati Uniti orientali (Virginia settentrionale) (Fase 1), è possibile creare il secondo cluster negli Stati Uniti orientali (Ohio) avviando contemporaneamente il processo di peering includendo l'ARN del primo cluster.

Example

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2","clusters": ["arn:aws:dsql:us-
east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Questo approccio combina i passaggi 2 e 4, ma è comunque necessario completare il passaggio 3 (aggiornamento del primo cluster con l'ARN del secondo cluster) per stabilire la relazione di peering. Una volta completati tutti i passaggi, entrambi i cluster passeranno attraverso gli stessi stati del processo standard: da IN ATTESA DI CONFIGURAZIONE a CREAZIONE IN CORSO e infine allo stato ATTIVO quando sono pronti per l'uso.

Eliminazione di cluster multi-Regione

Per eliminare un cluster multi-Regione, è necessario completare due passaggi.

1. Disattivare la protezione dall'eliminazione per ogni cluster.
2. Eliminare ogni cluster peered separatamente nelle rispettive Regione AWS

Aggiornare ed eliminare il cluster negli Stati Uniti orientali (Virginia settentrionale)

1. Disattivare la protezione da eliminazione utilizzando il comando `update-cluster`.

   ```
   aws dsql update-cluster \
   ```

```
  --region us-east-1 \
  --identifier 'foo0bar1baz2quux3quuxquux4' \
  --no-deletion-protection-enabled
```

2. Eliminare il cluster utilizzando il comando `delete-cluster`.

```
aws dsql delete-cluster \
  --region us-east-1 \
  --identifier 'foo0bar1baz2quux3quuxquux4'
```

Il comando restituisce la seguente risposta.

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_DELETE",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Il cluster passa allo stato `PENDING_DELETE`. L'eliminazione non è completa finché non si elimina il cluster peer negli Stati Uniti orientali (Ohio).

Aggiornare ed eliminare il cluster negli Stati Uniti orientali (Ohio)

1. Disattivare la protezione da eliminazione utilizzando il comando `update-cluster`.

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quux4quuux' \
--no-deletion-protection-enabled
```

2. Eliminare il cluster utilizzando il comando `delete-cluster`.

```
aws dsql delete-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5'
```

Il comando restituisce la seguente risposta:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/
foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_DELETE",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Il cluster passa allo stato `PENDING_DELETE`. Dopo alcuni secondi, il sistema passa automaticamente allo stato `DELETING` entrambi i cluster peered dopo la convalida.

Configurazione dei cluster Aurora DSQL utilizzando AWS CloudFormation

È possibile utilizzare la stessa risorsa CloudFormation `AWS::DSQL::Cluster` per distribuire e gestire cluster Aurora DSQL a Regione singola e multi-Regione.

Consultare Guida di riferimento ai tipi di risorse di Amazon Aurora DSQL per maggiori informazioni su come creare, modificare e gestire i cluster utilizzando la risorsa `AWS::DSQL::Cluster`.

Creazione della configurazione iniziale del cluster

Innanzitutto, creare un modello AWS CloudFormation per definire il cluster multi-Regione:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
```

Creare stack in entrambe le regioni utilizzando i seguenti comandi della CLI di AWS:

```
aws cloudformation create-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation create-stack --region us-east-1 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

Individuazione di identificatori di cluster

Recuperare gli ID delle risorse fisiche dei cluster:

```
aws cloudformation describe-stack-resources -region us-east-2 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "auabudrks5jwh4mjt6o5xxhr4y"
]
```

```
aws cloudformation describe-stack-resources -region us-east-1 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "imabudrfon4p2z3nv2jo4rlajm"
]
```

Aggiornamento della configurazione di un cluster

Aggiornare il modello AWS CloudFormation per includere entrambi gli ARN del cluster:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
        Clusters:
        - arn:aws:dsql:us-east-2:123456789012:cluster/auabudrks5jwh4mjt6o5xxhr4y
```

```
        - arn:aws:dsql:us-east-1:123456789012:cluster/imabudrfon4p2z3nv2jo4rlajm
```

Applicare la configurazione aggiornata a entrambe le regioni:

```
aws cloudformation update-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation update-stack --region us-east-1 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

Ciclo di vita del cluster Aurora DSQL

Comprendere il ciclo di vita del cluster Aurora DSQL aiuta a gestire i cluster in modo efficace. Questo capitolo tratta le definizioni dello stato dei cluster e la funzionalità di scalabilità a zero che ottimizza i costi.

Definizione dello stato del cluster Aurora DSQL

Lo stato del cluster Aurora DSQL fornisce informazioni critiche sullo stato e la connettività del cluster. È possibile visualizzare lo stato dei cluster e delle istanze del cluster utilizzando l' Console di gestione AWS API SQL di Aurora o Aurora. AWS CLI

La tabella seguente descrive ogni stato possibile per un cluster Aurora DSQL e il significato di ogni stato.

Status	Description
Creazione in corso	Aurora DSQL sta tentando di creare o configurare risorse per il cluster. Qualsiasi tentativo di connessione fallirà mentre un cluster si trova in questo stato.
Attivo	Il cluster è operativo e pronto per l'uso.
Sospeso	Un cluster diventa inattivo quando rimane inattivo abbastanza a lungo da consentire ad Aurora DSQL di ridimensionare le risorse in esecuzione per ridurre capacità e costi. Quando ci si connette a un cluster in stato sospeso, Aurora DSQL riporta il cluster allo stato Attivo.

Status	Description
Inattivo	Un cluster inattivo diventa inattivo quando non vi è stata alcuna attività sul cluster per un periodo prolungato. In questo stato di sospensione, le risorse in esecuzione vengono ridimensionate a zero mentre i dati vengono preservat i. Quando si tenta di connettersi a un cluster inattivo, Aurora DSQL riporta automaticamente il cluster allo stato Attivo. Il tempo di ripristino dipende dalla dimensione del cluster.
Aggiornam ento in corso	Un cluster passa allo stato di Aggiornamento in corso quando si apportano modifiche alla configurazione del cluster.
Eliminazione in corso	Un cluster passa allo stato Eliminazione in corso quando si invia una richiesta di eliminazione.
Eliminato	Il cluster è stato eliminato correttamente.
Non riuscito	Aurora DSQL non è riuscita a creare il cluster perché ha rilevato un errore.
In attesa di configurazione	Solo per cluster multi-Regione. Un cluster multi-Regione passa allo stato di In attesa di configurazione quando si crea un cluster multi-Regione nella prima Regione con una Regione testimone. La creazione del cluster viene sospesa fino a quando non si crea un altro cluster in una Regione secondaria e si esegue il peering dei due cluster.
In attesa di eliminazione	Solo per cluster multi-Regione. Un cluster multi-Regione passa allo stato In attesa di eliminazione quando si elimina uno dei suoi cluster. Il cluster passa allo stato di Eliminazione in corso dopo l'eliminazione dell'ultimo cluster in peering.

Utilizzo di cluster inattivi e inattivi

Quando Aurora DSQL non rileva alcuna attività di connessione su un cluster per un certo periodo di tempo, passa allo stato di inattività, riducendo le risorse in esecuzione per ridurre al minimo capacità e costi. Se l'attività di connessione rimane assente per un periodo prolungato, il cluster Idle passa automaticamente allo stato Inattivo, in cui le risorse in esecuzione vengono ridimensionate a zero mentre i dati vengono preservati.

Per riprendere le normali operazioni, è sufficiente connettersi al cluster come di consueto. Quando ci si connette correttamente al cluster, Aurora Aurora DSQL passa automaticamente il cluster allo stato Attivo.

> **Note**
>
> Il primo tentativo di connessione a un cluster inattivo o inattivo sarà più lento del solito.

Operazioni che richiedono lo stato attivo del cluster

Alcune operazioni richiedono che il cluster sia in uno stato Attivo. Per eseguire queste operazioni su un cluster inattivo o inattivo, è necessario riportare il cluster ad Active semplicemente connettendosi al cluster.

Operazioni di backup

L'esecuzione di un backup richiede uno stato attivo del cluster. Se il cluster è inattivo o inattivo, i backup falliranno con il seguente errore:

```
"Error": {
    "Code": "FailedPrecondition",
    "Message": "Cluster 'cluster-id' is in state 'IDLE' and cannot be backed up.
    In order to take a backup of your cluster, it must be in Active state. Please
    connect to your cluster to transition it to Active to perform the backup."
}
```

Per procedere con un backup:

1. Connect al cluster utilizzando il client di database preferito o la console Aurora DSQL per riattivarlo.
2. Attendi la transizione automatica allo stato Attivo.
3. Avvia il backup una volta che il cluster è completamente operativo.

> **Note**
>
> I backup esistenti eseguiti prima della transizione del cluster a inattivo o inattivo rimangono validi e inalterati. I nuovi tentativi di backup sul cluster falliranno finché il cluster non sarà connesso per la riattivazione automatica.

Visualizzazione dello stato del cluster Aurora DSQL

Per visualizzare lo stato del cluster, usa l'API SQL Console di gestione AWS di Aurora o Aurora. AWS CLI

Console

Segui questi passaggi per visualizzare lo stato del cluster nella Console di gestione AWS:

Visualizzazione dello stato di un cluster mediante la console

1. Apri la console di Aurora DSQL all'indirizzo https://console.aws.amazon.com/dsql.
2. Nel pannello di navigazione, scegliere Cluster.
3. Visualizza lo stato di ogni cluster nel pannello di controllo.

AWS CLI

Usa il AWS CLI comando seguente per controllare lo stato di un singolo cluster.

```
aws dsql get-cluster --identifier cluster-id --query status --output text
```

Per visualizzare lo stato di tutti i cluster, esegui il seguente comando.

```
for id in $(aws dsql list-clusters --query 'clusters[*].identifier' --output text); do

  cluster_status=$(aws dsql get-cluster --identifier "$id" --query 'status' --output
 text)
  echo "$id    $cluster_status"
done
```

Questo output di esempio mostra due cluster attivi e un cluster che è in fase di eliminazione.

```
aaabbb2bkx555xa7p42qd5cdef    ACTIVE
abcde123efghi77t35abcdefgh    ACTIVE
12abc6lqasc5bbbbbbbbbbbbbb    DELETING
```

Programmazione con Aurora DSQL

Aurora DSQL offre i seguenti strumenti per gestire le risorse Aurora DSQL a livello di programmazione.

AWS Command Line Interface(AWS CLI)

Puoi creare e gestire le tue risorse utilizzando la shell AWS CLI a riga di comando. AWS CLIFornisce l'accesso diretto al modulo APIs Servizi AWS, come Aurora DSQL. Per la sintassi e gli esempi dei comandi per Aurora DSQL, consulta dsql nella Guida di riferimento ai comandi di AWS CLI.

AWSkit di sviluppo software () SDKs

AWSfornisce SDKs molte tecnologie e linguaggi di programmazione popolari. Semplificano le chiamate Servizi AWS dall'interno delle applicazioni in quel linguaggio o tecnologia. Per ulteriori informazioni al riguardo SDKs, consulta Strumenti per lo sviluppo e la gestione di applicazioni su AWS.

API di Aurora DSQL

Questa API è un'altra interfaccia di programmazione per Aurora DSQL. Quando si utilizza questa API, è necessario formattare correttamente ogni richiesta HTTPS e aggiungere una firma digitale valida a ogni richiesta. Per ulteriori informazioni, consulta Guida di riferimento alle API.

CloudFormation

AWS::DSQL::ClusterSi tratta di una CloudFormation risorsa che consente di creare e gestire i cluster Aurora DSQL come parte dell'infrastruttura sotto forma di codice. CloudFormationti aiuta a definire l'intero AWS ambiente in codice, semplificando il provisioning, l'aggiornamento e la replica dell'infrastruttura in modo coerente e affidabile.

Quando utilizzi la AWS::DSQL::Cluster risorsa nei tuoi CloudFormation modelli, puoi effettuare il provisioning dichiarativo dei cluster Aurora DSQL insieme alle altre risorse cloud. Questo aiuta a garantire che l'infrastruttura dei dati venga distribuita e gestita insieme al resto dello stack di applicazioni.

Connettori per Aurora DSQL

Aurora DSQL fornisce connettori specializzati che estendono i driver di database esistenti per consentire l'autenticazione e l'integrazione IAM senza interruzioni con i servizi. AWS Questi

connettori sono progettati per funzionare con i linguaggi e i framework di programmazione più diffusi, pur mantenendo la compatibilità con i flussi di lavoro PostgreSQL esistenti.

Sono previsti connettori aggiuntivi per le versioni future. Per le informazioni più recenti sulla disponibilità dei connettori, consulta l'archivio di esempi per Aurora DSQL.

Connessione ai cluster Aurora DSQL con un connettore JDBC

L'Aurora DSQL Connector per JDBC è progettato come un plug-in di autenticazione che estende la funzionalità del driver JDBC PostgreSQL per consentire alle applicazioni di autenticarsi con Aurora DSQL utilizzando le credenziali IAM. Il connettore non si connette direttamente al database, ma fornisce un'autenticazione IAM senza semplificata in aggiunta al driver JDBC PostgreSQL sottostante.

Il connettore Aurora DSQL per JDBC è progettato per funzionare con il driver JDBC PostgreSQL e offre una perfetta integrazione con i requisiti di autenticazione IAM di Aurora DSQL.

Oltre al driver JDBC PostgreSQL, il connettore Aurora DSQL per JDBC consente l'autenticazione basata su IAM per Aurora DSQL. Introduce una profonda integrazione con servizi di autenticazione come (IAM). AWS AWS Identity and Access Management

Informazioni sul connettore

Aurora DSQL è un servizio di database SQL distribuito che offre disponibilità e scalabilità elevate per applicazioni compatibili con PostgreSQL. Aurora DSQL richiede l'autenticazione basata su IAM con token a tempo limitato che i driver JDBC esistenti non supportano nativamente.

L'idea principale alla base di Aurora DSQL Connector per JDBC è aggiungere un livello di autenticazione al driver JDBC PostgreSQL che gestisce la generazione di token IAM, permettendo agli utenti di connettersi ad Aurora DSQL senza modificare i flussi di lavoro JDBC esistenti.

Che cos'è l'autenticazione di Aurora DSQL?

In Aurora DSQL, l'autenticazione prevede:

- Autenticazione IAM: tutte le connessioni utilizzano l'autenticazione basata su IAM con token a tempo limitato
- Generazione di token: AWS i token di autenticazione vengono generati utilizzando credenziali e hanno una durata configurabile

Il connettore Aurora DSQL per JDBC è progettato per comprendere questi requisiti e generare automaticamente token di autenticazione IAM quando si stabiliscono connessioni.

Vantaggi del connettore Aurora DSQL per JDBC

Sebbene Aurora DSQL fornisca un'interfaccia compatibile con PostgreSQL, i driver PostgreSQL esistenti attualmente non supportano i requisiti di autenticazione IAM di Aurora DSQL. L'Aurora DSQL Connector per JDBC consente ai clienti di continuare a utilizzare i flussi di lavoro PostgreSQL esistenti abilitando al contempo l'autenticazione IAM tramite:

- Generazione automatica di token: i token IAM vengono generati automaticamente utilizzando le credenziali AWS
- Integrazione perfetta: funziona con i modelli di connessione JDBC esistenti
- AWS Supporto credenziali: supporta vari provider di AWS credenziali (predefiniti, basati sul profilo, ecc.)

Utilizzo del connettore Aurora DSQL per JDBC con pool di connessioni

Il connettore Aurora DSQL per JDBC funziona con librerie di connection pooling come HikariCP. Il connettore gestisce la generazione di token IAM durante la creazione della connessione, consentendo ai pool di connessioni di funzionare normalmente.

Funzionalità principali

Generazione automatica di token

I token IAM vengono generati automaticamente utilizzando le credenziali. AWS

Integrazione perfetta

Funziona con gli schemi di connessione JDBC esistenti senza richiedere modifiche al flusso di lavoro.

AWS Supporto per le credenziali

Supporta vari provider di AWS credenziali (predefiniti, basati sul profilo, ecc.).

Compatibilità della gestione di pool di connessioni

Funziona perfettamente con le librerie di gestione dei pool di connessioni come HikariCP.

Prerequisiti

Prima di cominciare, assicurarsi che i seguenti requisiti preliminari siano soddisfatti:

- Creazione di un cluster in Aurora DSQL.
- Installazione di Java Development Kit (JDK). Verifica della disponibilità della versione 17 o superiore.
- Configurazione delle autorizzazioni IAM appropriate per consentire all'applicazione di connettersi ad Aurora DSQL.
- AWS credenziali configurate (tramite AWS CLI variabili di ambiente o ruoli IAM).

Utilizzo del connettore Aurora DSQL per JDBC

Per utilizzare il connettore Aurora DSQL per JDBC nella tua applicazione Java, procedi nel seguente modo:

1. Aggiungere la seguente dipendenza Maven al progetto:

```
<dependencies>
    <!-- Aurora DSQL Connector for JDBC -->
    <dependency>
        <groupId>software.amazon.dsql</groupId>
        <artifactId>aurora-dsql-jdbc-connector</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

   Per i progetti Gradle, aggiungere questa dipendenza:

```
implementation("software.amazon.dsql:aurora-dsql-jdbc-connector:1.0.0")
```

2. Crea una connessione di base al tuo cluster Aurora DSQL utilizzando il formato del connettore DSQL AWS PostgreSQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
public class DsqlJdbcConnectorExample {
    public static void main(String[] args) {
        // Using AWS DSQL PostgreSQL Connector prefix
        String jdbcUrl = "jdbc:aws-dsql:postgresql://your-cluster.dsql.us-
east-1.on.aws/postgres?user=admin";

        try (Connection connection = DriverManager.getConnection(jdbcUrl)) {
            // Use the connection
            try (Statement statement = connection.createStatement()) {
                // Create a table
                statement.execute("CREATE TABLE IF NOT EXISTS test_table (id UUID
 PRIMARY KEY DEFAULT gen_random_uuid(), name VARCHAR(100))");

                // Insert data
                statement.execute("INSERT INTO test_table (name) VALUES ('Test
 Name')");

                // Query data
                try (ResultSet resultSet = statement.executeQuery("SELECT * FROM
 test_table")) {
                    while (resultSet.next()) {
                        System.out.println("ID: " + resultSet.getInt("id") + ",
 Name: " + resultSet.getString("name"));
                    }
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Proprietà della configurazione

Il connettore Aurora DSQL per JDBC supporta le seguenti proprietà di connessione:

user

Determina l'utente per la connessione e il metodo di generazione del token utilizzato. Ad esempio: `admin`

token-duration-secs

Durata in secondi della validità del token. Per maggiori informazioni sui limiti dei token, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.

profile

Utilizzato per creare un'istanza di una generazione di token ProfileCredentialsProvider for con il nome di profilo fornito.

region

AWS regione per le connessioni Aurora DSQL. Facoltativo. Se fornita, sostituirà la Regione estratta dall'URL.

database

Il nome del database a cui connettersi. Il valore predefinito è `postgres`.

Registrazione dei log

Abilita la registrazione per risolvere eventuali problemi che si potrebbero riscontrare durante l'utilizzo del connettore JDBC di Aurora DSQL.

Il connettore utilizza il sistema di registrazione dei log integrato (java.util.logging) di Java. È possibile configurare i livelli di registrazione creando un file `logging.properties`:

```
# Set root logger level to INFO for clean output
.level = INFO

# Show Aurora DSQL Connector for JDBC FINE logs for detailed debugging
software.amazon.dsql.level = FINE

# Console handler configuration
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Detailed formatter pattern with timestamp and logger name
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Esempi

Per esempi e casi d'uso più completi, consulta l'archivio Aurora DSQL Connector for JDBC

Connettore Aurora DSQL per Python

Il connettore Aurora DSQL per Python integra l'autenticazione IAM per connettere le applicazioni Python ai cluster DSQL di Amazon Aurora. Internamente, utilizza le librerie client psycopg, psycopg2 e asyncpg.

L'Aurora DSQL Connector per Python è progettato come un plug-in di autenticazione che estende la funzionalità delle librerie client psycopg, psycopg2 e asyncpg per consentire alle applicazioni di autenticarsi con Amazon Aurora DSQL utilizzando credenziali IAM. Il connettore non si connette direttamente al database ma fornisce un'autenticazione IAM senza interruzioni in aggiunta alle librerie client sottostanti.

Informazioni sul connettore

Amazon Aurora DSQL è un servizio di database SQL distribuito che offre disponibilità e scalabilità elevate per applicazioni compatibili con PostgreSQL. Aurora DSQL richiede l'autenticazione basata su IAM con token a tempo limitato che le librerie Python esistenti non supportano nativamente.

L'idea alla base di Aurora DSQL Connector for Python è quella di aggiungere un livello di autenticazione alle librerie client psycopg, psycopg2 e asyncpg che gestiscono la generazione di token IAM, permettendo agli utenti di connettersi ad Aurora DSQL senza modificare i flussi di lavoro esistenti.

Che cos'è l'autenticazione di Aurora DSQL?

In Aurora DSQL, l'autenticazione prevede:

- Autenticazione IAM: tutte le connessioni utilizzano l'autenticazione basata su IAM con token a tempo limitato
- Generazione di token: i token di autenticazione vengono generati utilizzando credenziali AWS e hanno una durata configurabile

Il connettore Aurora DSQL per Python è progettato per comprendere questi requisiti e generare automaticamente token di autenticazione IAM quando si stabiliscono connessioni.

Funzionalità

- Autenticazione IAM automatica: i token IAM vengono generati automaticamente utilizzando le credenziali AWS
- Basato su psycopg, psycopg2 e asyncpg, sfrutta le librerie client psycopg, psycopg2 e asyncpg

- Integrazione perfetta: funziona con i pattern di connessione psycopg, psycopg2 e asyncpg esistenti senza richiedere modifiche al flusso di lavoro
- Region Auto-Discovery: estrae la regione AWS dal nome host del cluster DSQL
- AWS Credentials Support: supporta vari provider di credenziali AWS (predefiniti, basati su profili, personalizzati)
- Compatibilità con il pool di connessioni: funziona con il pool di connessioni integrato psycopg, psycopg2 e asyncpg

Guida rapida di avvio

Requisiti

- Python 3.10 o versioni successive
- Accesso a un cluster Aurora DSQL
- Configurazione delle autorizzazioni IAM appropriate per consentire all'applicazione di connettersi ad Aurora DSQL.
- Credenziali AWS configurate (tramite CLI AWS, variabili di ambiente o ruoli IAM)

Installazione

```
pip install aurora-dsql-python-connector
```

Installa psycopg o psycopg2 o asyncpg separatamente

Il programma di installazione di Aurora DSQL Connector for Python non installa le librerie sottostanti. Devono essere installate separatamente, ad esempio:

```
# Install psycopg and psycopg pool
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
pip install psycopg2-binary
```

```
# Install asyncpg
pip install asyncpg
```

Nota:

È necessario installare solo la libreria necessaria. Pertanto, se il client intende utilizzare psycopg, deve essere installato solo psycopg. Se il client intende utilizzare psycopg2, è necessario installare solo psycopg2. Se il client intende utilizzare asyncpg, deve essere installato solo asyncpg.

Se il client ne ha bisogno di più di una, è necessario installare tutte le librerie necessarie.

Utilizzo di base

psicocopia

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
```

```
    import aurora_dsql_asyncpg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Usando just host

psicocopia

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

Utilizzando solo l'ID del cluster

psicocopia

```
    import aurora_dsql_psycopg as dsql
```

```
    conn = dsql.connect("your-cluster")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster")
```

Nota:

Nello scenario «using just cluster ID», viene utilizzata la regione precedentemente impostata sulla macchina, ad esempio:

```
aws configure set region us-east-1
```

Se la regione non è stata impostata o l'ID del cluster specificato si trova in una regione diversa, la connessione avrà esito negativo. Per farlo funzionare, fornisci la regione come parametro come nell'esempio seguente:

psicocopia

```
    import aurora_dsql_psycopg as dsql

    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql
```

```
    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
            "region": "us-east-1",
    }

    conn = await dsql.connect("your-cluster", **config)
```

Stringa di connessione

psicocopia

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/
postgres?user=admin&token_duration_secs=15")
```

Configurazione avanzata

psicocopia

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql
```

```
    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Opzioni di configurazione

Opzione	Tipo	Campo obbligatorio	Descrizione
`host`	`string`	Sì	nome host o ID del cluster DSQL
`user`	`string`	No	Nome utente DSQL. Predefinito: admin
`dbname`	`string`	No	Nome del database. Predefinito: postgres
`region`	`string`	No	Regione AWS (rilevata automaticamente dal nome host se non fornita)
`port`	`int`	No	L'impostazione predefinita è 5432
`custom_credentials_provider`	`CredentialProvider`	No	Provider di credenziali AWS personalizzate
`profile`	`string`	No	Il nome del profilo IAM. Predefinito: predefinito.
`token_duration_secs`	`int`	No	Tempo di scadenza del token in secondi

Sono supportate anche tutte le opzioni di connessione standard delle librerie psycopg, psycopg2 e asyncpg sottostanti, ad eccezione dei parametri asyncpg krbsrvname e gsslib che non sono supportati da DSQL.

Utilizzo del connettore Aurora DSQL per Python con pool di connessioni

Il connettore Aurora DSQL per Python funziona con il pool di connessioni integrato psycopg, psycopg2 e asyncpg. Il connettore gestisce la generazione di token IAM durante la creazione della connessione, consentendo ai pool di connessioni di funzionare normalmente.

psicocopia

Per psycopg, il connettore implementa una classe di connessione denominata DSQLConnection che può essere passata direttamente a psycopg_pool. ConnectionPool costruttore. Per le operazioni asincrone, esiste anche una versione asincrona della classe denominata Connection. DSQLAsync

```
from psycopg_pool import ConnectionPool as PsycopgPool

...
pool = PsycopgPool(
    "",
    connection_class=dsql.DSQLConnection,
    kwargs=conn_params,
    min_size=2,
    max_size=8,
    max_lifetime=3300
)
```

Nota: configurazione max_lifetime di Connection

Il parametro max_lifetime deve essere impostato su meno di 3600 secondi (un'ora), poiché questa è la durata massima della connessione consentita dal database Aurora DSQL. L'impostazione di un valore max_lifetime inferiore consente al pool di connessioni di gestire in modo proattivo il riciclo delle connessioni, il che è più efficiente della gestione degli errori di timeout di connessione dal database.

psycopg2

Per psycopg2, il connettore fornisce una classe denominata DSQLThreaded ConnectionPool Aurora che eredita da psycopg2.pool. ThreadedConnectionPool. La DSQLThreaded ConnectionPool classe Aurora sovrascrive solo il metodo interno _connect. Il resto dell'implementazione è fornito da psycopg2.pool. ThreadedConnectionPool immutato.

```
    import aurora_dsql_psycopg2 as dsql

    pool = dsql.AuroraDSQLThreadedConnectionPool(
            minconn=2,
            maxconn=8,
            **conn_params,
    )
```

asincrono

Per asyncpg, il connettore fornisce una funzione create_pool che restituisce un'istanza di Asyncpg.Pool.

```
    import asyncio
    import os

    import aurora_dsql_asyncpg as dsql

    pool_params = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'user': "admin",
        "min_size": 2,
        "max_size": 5,
    }

    pool = await dsql.create_pool(**pool_params)
```

Autenticazione

Il connettore gestisce automaticamente l'autenticazione DSQL generando token utilizzando il generatore di token client DSQL. Se la regione AWS non viene fornita, verrà analizzata automaticamente a partire dal nome host fornito.

Per ulteriori informazioni sull'autenticazione in Aurora DSQL, consulta la guida per l'utente.

Amministratore e utenti regolari

- Gli utenti nominati utilizzano `"admin"` automaticamente i token di autenticazione dell'amministratore
- Tutti gli altri utenti utilizzano token di autenticazione non amministrativi
- I token vengono generati dinamicamente per ogni connessione

Esempi

Per un codice di esempio completo, fate riferimento agli esempi indicati nelle sezioni seguenti. Per istruzioni su come eseguire gli esempi, fate riferimento ai file READMDE di esempio.

psicocopia

Esempi README

Description	Esempi
Utilizzo del connettore Aurora DSQL per Python per connessioni di base	Esempio di connessione di base
Utilizzo del connettore Aurora DSQL per Python per connessioni asincrone di base	Esempio di connessione asincrona di base
Utilizzo del connettore Aurora DSQL per Python con pool di connessioni	Esempio di connessione di base con pool di connessioni
	Esempio di connessioni simultanee con pool di connessioni
Utilizzo del connettore Aurora DSQL per Python con pool di connessioni asincrone	Esempio di connessione di base con pool di connessioni asincrone

psycopg2

Esempi README

Description	Esempi
Utilizzo del connettore Aurora DSQL per Python per connessioni di base	Esempio di connessione di base
Utilizzo del connettore Aurora DSQL per Python con pool di connessioni	Esempio di connessione di base con pool di connessioni
	Esempio di connessioni simultanee con pool di connessioni

asyncpg

Esempi README

Description	Esempi
Utilizzo del connettore Aurora DSQL per Python per connessioni di base	Esempio di connessione di base
Utilizzo del connettore Aurora DSQL per Python con pool di connessioni	Esempio di connessione di base con pool di connessioni
	Esempio di connessioni simultanee con pool di connessioni

Connettori Aurora DSQL per Node.js

Aurora DSQL Connector per node-postgres e Aurora DSQL Connector per Postgres.js sono plugin di autenticazione che estendono la funzionalità dei client node-postgres e Postgres.js per consentire alle applicazioni di autenticarsi con Aurora DSQL utilizzando le credenziali IAM.

Connettore Aurora DSQL per node-postgres

Il connettore Aurora DSQL per node-postgres è un connettore Node.js basato su node-postgres che integra l'autenticazione IAM per connettere le applicazioni ai cluster DSQL di Amazon Aurora. JavaScript/TypeScript

L'Aurora DSQL Connector è progettato come un plug-in di autenticazione che estende la funzionalità del client e del pool di node-postgres per consentire alle applicazioni di autenticarsi con Amazon Aurora DSQL utilizzando credenziali IAM.

Informazioni sul connettore

Amazon Aurora DSQL è un database distribuito nativo del cloud con compatibilità PostgreSQL. Sebbene richieda l'autenticazione IAM e token temporizzati, i driver di database Node.js tradizionali non dispongono di questo supporto integrato.

Il connettore Aurora DSQL per node-postgres colma questa lacuna implementando un middleware di autenticazione che funziona perfettamente con node-postgres. Questo approccio consente agli sviluppatori di mantenere il codice node-postgres esistente ottenendo al contempo un accesso sicuro basato su IAM ai cluster Aurora DSQL attraverso la gestione automatizzata dei token.

Che cos'è l'autenticazione di Aurora DSQL?

In Aurora DSQL, l'autenticazione prevede:

- Autenticazione IAM: tutte le connessioni utilizzano l'autenticazione basata su IAM con token a tempo limitato
- Generazione di token: i token di autenticazione vengono generati utilizzando credenziali AWS e hanno una durata configurabile

Il connettore Aurora DSQL per node-postgres è progettato per comprendere questi requisiti e generare automaticamente token di autenticazione IAM quando si stabiliscono connessioni.

Funzionalità

- Autenticazione IAM automatica: gestisce la generazione e l'aggiornamento dei token DSQL
- Basato su node-postgres: sfrutta il popolare client PostgreSQL per Node.js
- Integrazione perfetta: funziona con i modelli di connessione node-postgres esistenti
- Region Auto-Discovery: estrae la regione AWS dal nome host del cluster DSQL

- Full TypeScript Support - Fornisce una sicurezza completa
- AWS Credentials Support: supporta vari provider di credenziali AWS (predefiniti, basati su profili, personalizzati)
- Compatibilità con il pool di connessioni: funziona perfettamente con il pool di connessioni integrato

Applicazione di esempio

Nell'esempio è inclusa un'applicazione di esempio che mostra come utilizzare Aurora DSQL Connector per node-postgres. Per eseguire l'esempio incluso, fai riferimento all'esempio README.

Guida rapida all'avvio

Requisiti

- Node.js 20+
- Accesso a un cluster Aurora DSQL
- Configurazione delle autorizzazioni IAM appropriate per consentire all'applicazione di connettersi ad Aurora DSQL.
- Credenziali AWS configurate (tramite CLI AWS, variabili di ambiente o ruoli IAM)

Installazione

```
npm install @aws/aurora-dsql-node-postgres-connector
```

Dipendenze tra pari

```
npm install @aws-sdk/credential-providers @aws-sdk/dsql-signer pg tsx
npm install --save-dev @types/pg
```

Utilizzo

Connessione client

```
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
```

```
});
await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Connessione al pool

```
import { AuroraDSQLPool } from "@aws/aurora-dsql-node-postgres-connector";

const pool = new AuroraDSQLPool({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
  max: 3,
  idleTimeoutMillis: 60000,
});

const result = await pool.query("SELECT NOW()");
```

Utilizzo avanzato

```
import { fromNodeProviderChain } from "@aws-sdk/credential-providers";
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "example.dsql.us-east-1.on.aws",
  user: "admin",
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
});

await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Opzioni di configurazione

Opzione	Tipo	Car obb rio	Descrizione
`host`	`string`	Sì	Nome host del cluster DSQL

Opzione	Tipo	Car obb rio	Descrizione
`username`	`string`	Sì	nome utente DSQL
`database`	`string`	No	Nome del database
`region`	`string`	No	Regione AWS (rilevata automatic amente dal nome host se non fornita)
`port`	`number`	No	L'impostazione predefinita è 5432
`customCre dentialsP rovider`	`AwsCredentialIdent ity / AwsCredentialIdent ityProvider`	No	Provider di credenziali AWS personalizzate
`profile`	`string`	No	Il nome del profilo IAM. L'imposta zione predefinita è «default»
`tokenDura tionSecs`	`number`	No	Tempo di scadenza del token in secondi

Tutti gli altri parametri di Client/Pool sono supportati.

Autenticazione

Il connettore gestisce automaticamente l'autenticazione DSQL generando token utilizzando il generatore di token client DSQL. Se la regione AWS non viene fornita, verrà analizzata automaticamente a partire dal nome host fornito.

Per ulteriori informazioni sull'autenticazione in Aurora DSQL, consulta la guida per l'utente.

Amministratore e utenti regolari

- Gli utenti denominati «admin» utilizzano automaticamente i token di autenticazione dell'amministratore
- Tutti gli altri utenti utilizzano token di autenticazione regolari
- I token vengono generati dinamicamente per ogni connessione

Connettore Aurora DSQL per Postgres.js

Il connettore Aurora DSQL per Postgres.js è un connettore Node.js basato su Postgres.js che integra l'autenticazione IAM per connettere le applicazioni ai cluster DSQL di JavaScript Amazon Aurora.

Il connettore Aurora DSQL per Postgres.js è progettato come un plug-in di autenticazione che estende la funzionalità del client Postgres.js per consentire alle applicazioni di autenticarsi con Amazon Aurora DSQL utilizzando credenziali IAM. Il connettore non si connette direttamente al database, ma fornisce un'autenticazione IAM senza interruzioni in aggiunta al driver Postgres.js sottostante.

Informazioni sul connettore

Amazon Aurora DSQL è un servizio di database SQL distribuito che offre disponibilità e scalabilità elevate per applicazioni compatibili con PostgreSQL. Aurora DSQL richiede l'autenticazione basata su IAM con token a tempo limitato che i driver Node.js esistenti non supportano in modo nativo.

L'idea alla base del connettore Aurora DSQL per Postgres.js consiste nell'aggiungere un livello di autenticazione al client Postgres.js che gestisce la generazione di token IAM, permettendo agli utenti di connettersi ad Aurora DSQL senza modificare i flussi di lavoro Postgres.js esistenti.

Il connettore Aurora DSQL per Postgres.js funziona con la maggior parte delle versioni di Postgres.js. Gli utenti forniscono la propria versione installando direttamente Postgres.js.

Che cos'è l'autenticazione di Aurora DSQL?

In Aurora DSQL, l'autenticazione prevede:

- Autenticazione IAM: tutte le connessioni utilizzano l'autenticazione basata su IAM con token a tempo limitato
- Generazione di token: i token di autenticazione vengono generati utilizzando credenziali AWS e hanno una durata configurabile

Il connettore Aurora DSQL per Postgres.js è progettato per comprendere questi requisiti e generare automaticamente token di autenticazione IAM quando si stabiliscono connessioni.

Funzionalità

- Autenticazione IAM automatica: gestisce la generazione e l'aggiornamento dei token DSQL
- Basato su Postgres.js: sfrutta il veloce client PostgreSQL per Node.js
- Integrazione perfetta: funziona con i modelli di connessione Postgres.js esistenti

- Region Auto-Discovery: estrae la regione AWS dal nome host del cluster DSQL
- Full TypeScript Support - Fornisce una sicurezza completa
- AWS Credentials Support: supporta vari provider di credenziali AWS (predefiniti, basati su profili, personalizzati)
- Compatibilità con il pool di connessioni: funziona perfettamente con il pool di connessioni integrato di Postgres.js

Guida rapida di avvio

Requisiti

- Node.js 20+
- Accesso a un cluster Aurora DSQL
- Configurazione delle autorizzazioni IAM appropriate per consentire all'applicazione di connettersi ad Aurora DSQL.
- Credenziali AWS configurate (tramite CLI AWS, variabili di ambiente o ruoli IAM)

Installazione

```
npm install @aws/aurora-dsql-postgresjs-connector
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Utilizzo di base

```
import { auroraDSQLPostgres } from '@aws/aurora-dsql-postgresjs-connector';

const sql = auroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  username: 'admin',

});

// Execute queries
const users = await sql`SELECT * FROM users WHERE age > ${25}`;
console.log(users);

// Clean up
await sql.end();
```

Utilizzo dell'ID del cluster anziché dell'host

```
const sql = auroraDSQLPostgres({
  host: 'your-cluster-id',
  region: 'us-east-1',
  username: 'admin',

});
```

Stringa di connessione

```
const sql = AuroraDSQLPostgres(
  'postgres://admin@your-cluster.dsql.us-east-1.on.aws'
);

const result = await sql`SELECT current_timestamp`;
```

Configurazione avanzata

```
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';

const sql = AuroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  database: 'postgres',
  username: 'admin',
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
  tokenDurationSecs: 3600,                            // Token expiration (seconds)

  // Standard Postgres.js options
  max: 20,                                // Connection pool size
  ssl: { rejectUnauthorized: false }    // SSL configuration
});
```

Opzioni di configurazione

Opzione	Tipo	Campo obbligatorio	Descrizione
`host`	`string`	Sì	Nome host o ID del cluster DSQL

Opzione	Tipo	Campo obbligatorio	Descrizione
`database`	`string?`	No	Nome del database
`username`	`string?`	No	Nome utente del database (utilizza admin se non fornito)
`region`	`string?`	No	Regione AWS (rilevata automaticamente dal nome host se non fornita)
`customCredentialsProvider`	`AwsCredentialIdentityProvider?`	No	Provider di credenziali AWS personalizzate
`tokenDurationSecs`	`number?`	No	Tempo di scadenza del token in secondi

Sono supportate anche tutte le opzioni standard di Postgres.js.

Autenticazione

Il connettore gestisce automaticamente l'autenticazione DSQL generando token utilizzando il generatore di token client DSQL. Se la regione AWS non viene fornita, verrà analizzata automaticamente a partire dal nome host fornito.

Per ulteriori informazioni sull'autenticazione in Aurora DSQL, consulta la guida per l'utente.

Amministratore e utenti regolari

- Gli utenti denominati «admin» utilizzano automaticamente i token di autenticazione dell'amministratore
- Tutti gli altri utenti utilizzano token di autenticazione regolari
- I token vengono generati dinamicamente per ogni connessione

Esempio di utilizzo

Un JavaScript esempio di utilizzo del connettore Aurora DSQL per Postgres.js è disponibile qui.

Accesso ad Aurora DSQL con client compatibili con PostgreSQL

Aurora DSQL utilizza il protocollo wire PostgreSQL. Puoi connetterti a PostgreSQL utilizzando una varietà di strumenti e client, AWS CloudShell come psql e. DBeaver DataGrip La tabella seguente riassume il modo in cui Aurora DSQL mappa i parametri di connessione PostgreSQL comuni:

PostgreSQL	Aurora DSQL	Note
Ruolo (noto anche come Utente o Gruppo)	Ruolo di database	Aurora DSQL crea un ruolo denominato `admin`. Quando si creano ruoli di database personalizzati, è necessario utilizzare il ruolo `admin` per associarli ai ruoli IAM per l'autenticazione durante la connessione al cluster. Per maggiori informazioni, consulta Configurazione dei ruoli di database personalizzati.
Host (noto anche come hostname o hostspec)	Endpoint del cluster	I cluster Aurora DSQL a Regione singola forniscono un unico endpoint gestito e reindirizzano automaticamente il traffico in caso di indisponibilità all'interno della Regione.
Porta	N/A: utilizzare l'impostazione predefinita `5432`	Questa è l'impostazione predefinita di PostgreSQL.
Database (dbname)	Utilizza `postgres`	Aurora DSQL crea questo database alla creazione del cluster.
Modalità SSL	SSL è sempre abilitato lato server	Aurora DSQL supporta la modalità SSL `require`. Le connessioni senza SSL vengono rifiutate da Aurora DSQL.
Password	Token di autenticazione	Aurora DSQL richiede token di autenticazione temporanei anziché password di lunga durata. Per ulteriori informazioni, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.

Durante la connessione, Aurora DSQL richiede un token di autenticazione IAM firmato al posto di una password tradizionale. Questi token temporanei vengono generati utilizzando la versione 4 di AWS Signature e vengono utilizzati solo durante la creazione della connessione. Una volta connessa, la sessione rimane attiva fino al termine o alla disconnessione del client.

Se si tenta di aprire una nuova sessione con un token scaduto, la richiesta di connessione fallisce e deve essere generato un nuovo token. Per ulteriori informazioni, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.

Accedi ad Aurora DSQL utilizzando client SQL

Aurora DSQL supporta più client compatibili con PostgreSQL per la connessione al cluster. Le sezioni seguenti descrivono come connettersi utilizzando PostgreSQL AWS CloudShell con o la riga di comando locale, oltre a strumenti basati su GUI come and. DBeaver JetBrains DataGrip Ogni client richiede un token di autenticazione valido come descritto nella sezione precedente.

Argomenti

- AWS CloudShellDa utilizzare per accedere ad Aurora DSQL con il terminale interattivo PostgreSQL (psql)
- Usa la CLI locale per accedere ad Aurora DSQL con il terminale interattivo PostgreSQL (psql)
- Utilizzare DBeaver per accedere ad Aurora DSQL
- Utilizzare JetBrains DataGrip per accedere ad Aurora DSQL
- risoluzione dei problemi

AWS CloudShellDa utilizzare per accedere ad Aurora DSQL con il terminale interattivo PostgreSQL (psql)

Utilizzare la seguente procedura per accedere ad Aurora DSQL con il terminale interattivo PostgreSQL da. AWS CloudShell Per ulteriori informazioni, consulta Cos'è. AWS CloudShell

Per connettersi utilizzando AWS CloudShell

1. Accedi alla console Aurora DSQL.
2. Scegli il cluster per il quale desideri aprire. CloudShell Se non è ancora stato creato un cluster, segui i passaggi indicati in Passo 1: Creazione di cluster Aurora DSQL a Regione singola o Creazione di un cluster multi-Regione.
3. Scegli Connetti con Query Editor, quindi scegli Connetti con CloudShell.

4. Scegli se vuoi connetterti come amministratore o con un ruolo di database personalizzato.
5. Scegli Avvia CloudShell e scegli Esegui nella finestra di CloudShell dialogo seguente.

Usa la CLI locale per accedere ad Aurora DSQL con il terminale interattivo PostgreSQL (psql)

Usa`psql`, un front-end basato su terminale per PostgreSQL, per inserire in modo interattivo le query, inviarle a PostgreSQL e visualizzare i risultati delle query.

> **Note**
>
> Per migliorare i tempi di risposta alle query, utilizzare il client PostgreSQL versione 17. Se usi la CLI in un ambiente diverso, assicurati di configurare manualmente Python versione 3.8+ e psql versione 14+.

Scarica il programma di installazione specifico per il tuo sistema operativo dalla pagina dei download di PostgreSQL. Per ulteriori informazioni*`psql`*, vedere Applicazioni client PostgreSQL sul sito Web PostgreSQL.

Se lo hai già AWS CLI installato, usa il seguente esempio per connetterti al tuo cluster.

```
# Aurora DSQL requires a valid IAM token as the password when connecting.
# Aurora DSQL provides tools for this and here we're using Python.
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token \
  --region us-east-1 \
  --expires-in 3600 \
  --hostname your_cluster_endpoint)

# Aurora DSQL requires SSL and will reject your connection without it.
export PGSSLMODE=require

# Connect with psql, which automatically uses the values set in PGPASSWORD and
 PGSSLMODE.
# Quiet mode suppresses unnecessary warnings and chatty responses but still outputs
 errors.
psql --quiet \
  --username admin \
  --dbname postgres \
```

```
  --host your_cluster_endpoint
```

Utilizzare DBeaver per accedere ad Aurora DSQL

DBeaver è un client SQL universale che può essere utilizzato per gestire qualsiasi database dotato di driver JDBC. È ampiamente utilizzato dagli sviluppatori e dagli amministratori di database grazie alle sue solide capacità di visualizzazione, modifica e gestione dei dati. Utilizzando le opzioni DBeaver di connettività cloud di Aurora, puoi connetterti DBeaver ad Aurora DSQL in modo nativo.

DBeaver I prodotti PRO (DBeaver Ultimate, DBeaver Team, CloudBeaver Enterprise e CloudBeaver AWS) offrono l'integrazione nativa con Aurora DSQL a partire dalla versione 25.3 tramite un tipo di connessione Aurora DSQL dedicato e tramite Cloud Explorer con un'esperienza di autenticazione senza interruzioni.

Se utilizzi una DBeaver versione diversa, inclusa la DBeaver Community Edition, la versione gratuita e open source, visita la pagina di download per le istruzioni di installazione. Usa la seguente procedura per connetterti al tuo cluster.

Per configurare una nuova connessione Aurora DSQL in DBeaver

1. Scegli Nuova connessione al database.
2. Nella finestra Nuova connessione al database, scegli PostgreSQL.
3. Nella scheda Impostazioni di connessione/Principale, scegli Connect by: Host e inserisci le seguenti informazioni:

   - Host: utilizza l'endpoint del cluster.

     Database: inserisci `postgres`

     Autenticazione: seleziona `Database Native`

     Nome utente: inserisci `admin`

     Password: genera un token di autenticazione. Copia il token generato e usarlo come password.
4. Ignora qualsiasi avviso e incolla il token di autenticazione nel campo Password. DBeaver

> **Note**
>
> È necessario impostare la modalità SSL nelle connessioni client. Aurora DSQL supporta `PGSSLMODE=require and PGSSLMODE=verify-full`. Aurora DSQL applica la comunicazione SSL lato server e rifiuta le connessioni non SSL. Per l'`verify-full`opzione è necessario installare i certificati SSL localmente. Per ulteriori informazioni, consulta i certificati SSL/TLS.

5. È necessario essere connessi al cluster e iniziare a eseguire istruzioni SQL.

> **Important**
>
> Le funzionalità amministrative fornite dai DBeaver database PostgreSQL (come Session Manager e Lock Manager) non si applicano ai database Aurora DSQL a causa della loro architettura unica. Sebbene accessibili, queste schermate non forniscono informazioni affidabili sullo stato o sullo stato del database.

Utilizzare JetBrains DataGrip per accedere ad Aurora DSQL

JetBrains DataGrip è un IDE multipiattaforma per lavorare con SQL e database, incluso PostgreSQL. DataGrip include una robusta GUI con un editor SQL intelligente. Per scaricare DataGrip, vai alla pagina di download sul JetBrainssito web.

Per configurare una nuova connessione Aurora DSQL in JetBrains DataGrip

1. Scegli Nuova origine dati e PostgreSQL.
2. Nella scheda Fonti dati/Generale, inserisci le seguenti informazioni:

   - Host: utilizza l'endpoint del cluster.

     Porta: Aurora DSQL utilizza l'impostazione predefinita di PostgreSQL `5432`

     Database: Aurora DSQL utilizza l'impostazione predefinita di PostgreSQL `postgres`

     Autenticazione: seleziona `User & Password` .

     Nome utente: inserisci `admin`.

Password: genera un token e incollalo in questo campo.

URL: non modificare questo campo. Verrà compilato automaticamente in base agli altri campi.

3. Password: forniscila generando un token di autenticazione. Copia l'output risultante dal generatore di token e incollalo nel campo password.

> **Note**
>
> È necessario impostare la modalità SSL nelle connessioni client. Aurora DSQL supporta `PGSSLMODE=require and PGSSLMODE=verify-full`. Aurora DSQL applica la comunicazione SSL lato server e rifiuta le connessioni non SSL. Per l'`verify-full`opzione è necessario installare i certificati SSL localmente. Per ulteriori informazioni, consulta i certificati SSL/TLS.

4. A questo punto è stabilita la connessione al cluster ed è possibile iniziare a eseguire le istruzioni SQL:

> **Important**
>
> Alcune viste fornite dai DataGrip database PostgreSQL (come Sessions) non si applicano ai database Aurora DSQL a causa della loro architettura unica. Sebbene accessibili, queste schermate non forniscono informazioni affidabili sulle sessioni effettive connesse al database.

risoluzione dei problemi

Scadenza delle credenziali di autenticazione per i client SQL

Le sessioni stabilite rimangono autenticate per un massimo di 1 ora o fino alla disconnessione esplicita o un timeout impostato lato client. Se è necessario stabilire nuove connessioni, è necessario generare e fornire un nuovo token di autenticazione nel campo Password della connessione.
Il tentativo di aprire una nuova sessione (ad esempio, per elencare nuove tabelle o aprire una nuova console SQL) impone un nuovo tentativo di autenticazione. Se il token di autenticazione configurato nelle impostazioni della Connessione non è più valido, la nuova sessione fallisce e tutte le sessioni aperte in precedenza vengono invalidate. Tienilo a mente quando scegli la durata del

token di autenticazione IAM con l'`expires-in`opzione, che può essere impostata su 15 minuti per impostazione predefinita e può essere impostata su un valore massimo di sette giorni.

Inoltre, consulta la sezione Risoluzione dei problemi della documentazione di Aurora DSQL.

Strumenti di connettività per i cluster Amazon Aurora DSQL

AWSfornisce vari strumenti per la connessione e l'utilizzo dei database Aurora DSQL. Questi includono driver di database, librerie ORM e adattatori specializzati che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio di programmazione preferito.

Driver del database

La tabella seguente mostra i driver di database disponibili per la connessione diretta ad Aurora DSQL.

Linguaggio di programmazione	Driver	Link al repository degli esempi
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	pgJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
Java	Connettore Aurora DSQL per JDBC	https://github.com/awslabs/aurora-dsql-jdbc-connector
JavaScript	nodo postgres	https://github.com/aws-samples/aurora-dsql-samples/-po

Linguaggio di programmazione	Driver	Link al repository degli esempi
		stgres tree/main/javascript/node
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	Psicocopia	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg 2	https://github.com/aws-samples/aurora-dsql-samples/2 tree/main/python/psycopg
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx

Librerie ORM (Object-Relational Mapping)

La tabella seguente mostra il codice di esempio per l'utilizzo di librerie ORM autonome con Aurora DSQL.

Linguaggio di programmazione	Libreria ORM	Link al repository degli esempi
Java	Ibernazione	https://github.com/awslabs/aurora-dsql-hibernate/tree/main/examples/pet-app clinica

Linguaggio di programmazione	Libreria ORM	Link al repository degli esempi
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/tree/main/examples/pet-app clinica
TypeScript	Sequelizza	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	Digitare ORM	https://github.com/aws-samples/aurora-dsql-samples/-orm tree/main/typescript/type

Adattatori e dialetti di Aurora DSQL

La tabella seguente mostra gli adattatori e i dialetti disponibili progettati specificamente per Aurora DSQL.

Linguaggio di programmazione	ORM/Framework	Link al repository
Java	Ibernazione	https://github.com/awslabs/aurora-dsql-hibernate/
Python	Django	https://github.com/awslabs/aurora-dsql-django/
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/

AI generativa per Aurora DSQL

Questa sezione fornisce istruzioni dettagliate su come utilizzare gli strumenti di intelligenza artificiale generativa con Aurora DSQL

Server MCP DSQL Aurora di AWS Labs

Un server MCP (Model Context Protocol) di AWS Labs per Aurora DSQL

Funzionalità

- Conversione di domande e comandi leggibili dall'uomo in query SQL strutturate compatibili con Postgres ed esecuzione sul database Aurora DSQL configurato.
- Sola lettura per impostazione predefinita, transazioni abilitate con `--allow-writes`
- Riutilizzo della connessione tra le richieste per migliorare le prestazioni
- Accesso integrato alla documentazione, alla ricerca e ai consigli sulle best practice di Aurora DSQL

Strumenti disponibili

Operazioni del database

- readonly_query - Esegui query SQL di sola lettura sul tuo cluster DSQL
- transact - Esegue operazioni di scrittura in una transazione (richiede) `--allow-writes`
- get_schema - Recupera le informazioni sullo schema della tabella

Documentazione e raccomandazioni

- dsql_search_documentation - Cerca nella documentazione DSQL di Aurora
  - Parametri: (obbligatorio), (opzionale) `search_phrase` `limit`
- dsql_read_documentation - Leggi pagine specifiche della documentazione DSQL
  - Parametri: `url` (richiesto), (opzionale), `start_index` (opzionale) `max_length`
- dsql_recommend: ottieni consigli sulle migliori pratiche DSQL
  - Parametri: (obbligatorio) `url`

Prerequisiti

1. Un account AWS con un cluster DSQL Aurora
2. Questo server MCP può essere eseguito solo localmente sullo stesso host del client LLM.
3. Configura le credenziali AWS con accesso ai servizi AWS
   - È necessario un account AWS con un ruolo che includa queste autorizzazioni:
     - `dsql:DbConnectAdmin`- Connect ai cluster DSQL come utente amministratore

  - `dsql:DbConnect`- Connect ai cluster DSQL con ruoli di database personalizzati (necessari solo se si utilizzano utenti non amministratori)
- Configura le credenziali AWS con le nostre `aws configure` variabili di ambiente

Installazione

Per la maggior parte degli strumenti, dovrebbe essere sufficiente aggiornare la configurazione seguendo le istruzioni di installazione predefinite.

Sono riportate istruzioni separate per Claude Code e Codex.

Installazione predefinita: aggiornamento del file di configurazione MCP pertinente

Uso di **uv**

1. Installa uv da Astral o dal README GitHub
2. Installa Python usando `uv python install 3.10`

Configurare il server MCP nella configurazione del client MCP (Finding the MCP Config File)

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "command": "uvx",
      "args": [
        "awslabs.aurora-dsql-mcp-server@latest",
        "--cluster_endpoint",
        "[your dsql cluster endpoint, e.g. abcdefghijklmnopqrst234567.dsql.us-
east-1.on.aws]",
        "--region",
        "[your dsql cluster region, e.g. us-east-1]",
        "--database_user",
        "[your dsql username, e.g. admin]",
        "--profile",
        "default"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR"
      },
      "disabled": false,
      "autoApprove": []
```

```
    }
  }
}
```

Installazione di Windows

Per gli utenti Windows, il formato di configurazione del server MCP è leggermente diverso:

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "disabled": false,
      "timeout": 60,
      "type": "stdio",
      "command": "uv",
      "args": [
        "tool",
        "run",
        "--from",
        "awslabs.aurora-dsql-mcp-server@latest",
        "awslabs.aurora-dsql-mcp-server.exe"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR",
        "AWS_PROFILE": "your-aws-profile",
        "AWS_REGION": "us-east-1"
      }
    }
  }
}
```

Individuazione del file di configurazione del client MCP

Per alcuni degli strumenti di sviluppo Agentic più comuni, è possibile trovare le configurazioni dei client MCP nei seguenti percorsi di file:

- Kiro:
  - Config utente: `~/.kiro/settings/mcp.json`
  - Config dell'area di lavoro: `/path/to/workspace/.kiro/settings/mcp.json`
- Claude Code: fai riferimento a Claude Code Installation per una guida dettagliata alla configurazione

  - Config utente: in `~/.claude.json` `"mcpServers"`
  - Config del progetto: `/path/to/project/.mcp.json`
  - Config locale: in `~/.claude.json` `"projects" -> "path/to/project" -> "mcpServers"`
- Cursore:
  - Globale: `~/.cursor/mcp.json`
  - Progetto: `/path/to/project/.cursor/mcp.json`
- Codice: `~/.codex/config.toml`
  - Ogni server MCP è configurato con una `[mcp_servers.<server-name>]` tabella nel file di configurazione. Consultate le istruzioni di installazione del Custom Codex
- Warp:
  - Modifica dei file: `~/.warp/mcp_settings.json`
  - Application Editor: `Settings > AI > Manage MCP Servers` e incolla json
- CLI per sviluppatori di Amazon Q: `~/.aws/amazonq/mcp.json`
- Cline: di solito un percorso VS Code annidato - `~/.vscode-server/path/to/cline_mcp_settings.json`

Codice Claude

Prerequisiti

Importante: la gestione del server MCP è disponibile solo tramite l'esperienza del terminale CLI di Claude Code, non la modalità pannello nativa di VS Code.

Installa prima la CLI di Claude Code seguendo la procedura di installazione nativa consigliata da Claude.

Scelta dell'ambito giusto

Claude Code offre 3 diversi ambiti: locale (predefinito), di progetto e utente e specifica quale ambito scegliere in base alla sensibilità delle credenziali e alla necessità di condividere. Per maggiori dettagli, consulta la documentazione di Claude Code sugli ambiti di installazione MCP.

1. I server con ambito locale rappresentano il livello di configurazione predefinito e sono memorizzati nel `~/.claude.json` percorso del progetto. Sono entrambi privati per te e accessibili solo all'interno della directory corrente del progetto. Questa è l'impostazione predefinita per la `scope` creazione di server MCP.
2. I server con ambito di progetto consentono la collaborazione in team pur rimanendo accessibili solo in una directory di progetto. I server con ambito di progetto aggiungono un `.mcp.json` file

nella directory principale del progetto. Questo file è progettato per essere archiviato nel controllo delle versioni, garantendo che tutti i membri del team abbiano accesso agli stessi strumenti e servizi MCP. Quando aggiungi un server con ambito di progetto, Claude Code crea o aggiorna automaticamente questo file con la struttura di configurazione appropriata.

3. I server con ambito utente vengono archiviati **`~/.claude.json`** e forniscono accessibilità tra progetti, rendendoli disponibili in tutti i progetti sul computer pur rimanendo privati per l'account utente.

Utilizzo della CLI di Claude (consigliato)

L'utilizzo di una sessione `claude` CLI interattiva consente una migliore esperienza di risoluzione dei problemi, quindi questo è il percorso consigliato.

```
claude mcp add amazon-aurora-dsql \
  --scope [one of local, project, or user] \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Risoluzione dei problemi: utilizzo di Claude Code con Bedrock su un altro account AWS

Se hai configurato Claude Code con un account o un profilo Bedrock AWS diverso dal profilo necessario per connetterti al tuo cluster dsql, dovrai fornire argomenti di ambiente aggiuntivi:

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Modifica diretta nel file di configurazione

Claude Code Richiede una denominazione alfanumerica, quindi ti consigliamo di assegnare un nome al tuo server:. `aurora-dsql-mcp-server`

Ambito locale

Aggiornamento `~/.claude.json` all'interno del campo specifico del progetto`mcpServers`:

```
{
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Ambito del progetto

Aggiornamento sul `/path/to/project/root/.mcp.json` campo: `mcpServers`

```
{
  "mcpServers": {}
}
```

Ambito dell'utente

Aggiornamento `~/.claude.json` all'interno del campo specifico del progetto: `mcpServers`

```
{
  "mcpServers": {}
}
```

Codex

Opzione 1: Codex CLI

Se hai installato la CLI Codex, puoi usare il comando codex mcp per configurare i tuoi server MCP.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Opzione 2: config.toml

Per un controllo più preciso sulle opzioni del server MCP, puoi modificare manualmente il file di configurazione. `~/.codex/config.toml` Ogni server MCP è configurato con una `[mcp_servers.<server-name>]` tabella nel file di configurazione.

```
[mcp_servers.amazon-aurora-dsql]
command = "uvx"
args = [
  "awslabs.aurora-dsql-mcp-server@latest",
  "--cluster_endpoint", "<DSQL_CLUSTER_ID>.dsql.<AWS_REGION>.on.aws",
  "--region", "<AWS_REGION>",
  "--database_user", "<DATABASE_USERNAME>"
]

[mcp_servers.amazon-aurora-dsql.env]
FASTMCP_LOG_LEVEL = "ERROR"
```

Verifica dell'installazione

Per Amazon Q Developer CLI, Kiro CLI, CLI/TUI, or Codex CLI/TUI Claude, `/mcp` esegui per vedere lo stato del server MCP.

Per l'IDE Kiro, puoi anche accedere alla `MCP SERVERS` scheda del pannello Kiro che mostra tutti i server MCP configurati e i relativi indicatori di stato della connessione.

Opzioni di configurazione del server

`--allow-writes`

Per impostazione predefinita, il server dsql mcp non consente operazioni di scrittura («modalità di sola lettura»). Qualsiasi richiamo dello strumento transact avrà esito negativo in questa modalità. Per utilizzare lo strumento di transazione, consenti le scritture passando il parametro. `--allow-writes`

Si consiglia di utilizzare l'accesso con privilegi minimi per la connessione a DSQL. Ad esempio, gli utenti devono utilizzare un ruolo di sola lettura quando possibile. La modalità di sola lettura prevede l'applicazione sul lato client del massimo impegno per rifiutare le mutazioni.

`--cluster_endpoint`

Questo è un parametro obbligatorio per specificare il cluster a cui connettersi. Questo dovrebbe essere l'endpoint completo del cluster, ad esempio `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`

`--database_user`

Questo è un parametro obbligatorio per specificare l'utente con cui connettersi. Ad esempio `admin` o `my_user`. Tieni presente che le credenziali AWS che stai utilizzando devono avere l'autorizzazione per accedere come tale utente. Per ulteriori informazioni sulla configurazione e l'utilizzo dei ruoli del database in DSQL, consulta Using database roles with IAM roles.

`--profile`

Puoi specificare il profilo aws da utilizzare per le tue credenziali. Nota che questo non è supportato per l'installazione dei docker.

È supportato anche l'utilizzo della variabile di `AWS_PROFILE` ambiente nella configurazione MCP:

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Se non viene fornito nessuno dei due, il server MCP utilizza per impostazione predefinita il profilo «predefinito» nel file di configurazione AWS.

`--region`

Questo è un parametro obbligatorio per specificare la regione del database DSQL.

`--knowledge-server`

Parametro opzionale per specificare l'endpoint del server MCP remoto per gli strumenti di conoscenza DSQL (ricerca della documentazione, lettura e consigli). Per impostazione predefinita, è preconfigurato.

Esempio:

```
--knowledge-server https://custom-knowledge-server.example.com
```

Nota: per motivi di sicurezza, utilizza solo endpoint Knowledge Server affidabili. Il server deve essere un endpoint HTTPS.

`--knowledge-timeout`

Parametro opzionale per specificare il timeout in secondi per le richieste al knowledge server.

Impostazione predefinita: `30.0`

Esempio:

```
--knowledge-timeout 60.0
```

Aumentate questo valore se si verificano dei timeout durante l'accesso alla documentazione su reti lente.

Inizia a usare Aurora DSQL Query Editor

Con Aurora DSQL Query Editor, puoi connetterti in modo sicuro ai tuoi cluster Aurora DSQL ed eseguire query SQL direttamente dalla console di gestione senza installare o configurare client esterni. AWS Fornisce uno spazio di lavoro intuitivo con evidenziazione della sintassi integrata, completamento automatico e assistenza intelligente del codice. È possibile esplorare rapidamente gli oggetti dello schema, sviluppare ed eseguire query SQL e visualizzare i risultati, il tutto all'interno di un'unica interfaccia.

Questo argomento illustra i passaggi per connettersi a un cluster, eseguire query, visualizzare i risultati ed esplorare funzionalità avanzate come i piani di esecuzione.

> **Note**
>
> L'editor di query è disponibile in tutte le regioni in cui è supportato Aurora DSQL. Per ulteriori informazioni, consulta AWSRegional Services.

Prerequisiti

Prima di iniziare, assicurati di soddisfare i seguenti requisiti:

- È disponibile almeno un cluster Aurora DSQL. Per ulteriori informazioni, consulta Passo 1: Creazione di cluster Aurora DSQL a Regione singola.
- L'endpoint del cluster è accessibile pubblicamente. L'editor di query attualmente non supporta i cluster con accesso pubblico bloccato da policy basate su risorse o cluster gestiti tramite endpoint VPC. Per ulteriori informazioni, consultare Blocco dell'accesso pubblico con politiche basate sulle risorse in Aurora DSQL e Gestione e connessione ai cluster SQL di Amazon Aurora tramite AWS PrivateLink.
- Il tuo utente o ruolo IAM dispone delle autorizzazioni necessarie per accedere e connettersi al cluster. Per ulteriori informazioni, consulta Utilizzo dei ruoli del database e dell'autenticazione IAM.

Lavorare con il Query Editor

Aprire l'editor di query

Per aprire l'editor di query

1. Apri la console Aurora DSQL.
2. Nel pannello di navigazione, scegliere Editor della query.

In alternativa, dalla pagina Cluster, seleziona il cluster su cui desideri interrogare e scegli Connect with Query editor per avviare direttamente l'editor.

> **Note**
>
> Lo stato di lavoro e di connessione non vengono salvati. Se si esce dalla console Aurora DSQL, si chiude la scheda del browser o si esce, le connessioni, il testo della query e i risultati vengono persi.

Connessione a un cluster

Per connettersi a un cluster

1. Se non esiste alcuna connessione al cluster, l'editor visualizza Nessun cluster è stato connesso. Scegli Connetti o seleziona + (Aggiungi) nel riquadro Cluster Explorer per connetterti a un cluster esistente.
2. (Facoltativo) Connettiti a più cluster o allo stesso cluster utilizzando ruoli diversi.

Esplora gli oggetti del cluster

Cluster Explorer visualizza tutte le connessioni al cluster disponibili e consente di sfogliare oggetti come database, schemi, tabelle e viste. Fornisce inoltre azioni comuni come Aggiorna, Crea tabella e altre opzioni specifiche del contesto.

Esecuzione di query

Per eseguire una query

1. Nel riquadro della scheda dell'editor di query, inserisci l'istruzione SQL. Esempio:

   ```
   SELECT * FROM public.orders LIMIT 10;
   ```

2. Verifica il contesto del cluster attivo visualizzato nella parte superiore destra della scheda di interrogazione. Ciò indica la connessione al cluster associata alla scheda di interrogazione corrente.
3. (Facoltativo) Utilizza il menu a discesa delle connessioni per esaminare tutte le connessioni disponibili o passare a un cluster diverso. La modifica della connessione aggiorna il punto in cui vengono eseguite le query in quella scheda.
4. Scegli Esegui per eseguire la query.

> **Note**
>
> Ogni query può restituire fino a 10.000 righe nel riquadro dei risultati. Per set di dati più grandi, perfeziona la query con filtri o limiti.

Rivedi i risultati e i piani di esecuzione

Dopo l'esecuzione della query, esaminate l'output nel pannello Risultati nella parte inferiore dell'editor. Per impostazione predefinita, a ogni esecuzione di query viene visualizzata la scheda Risultati (tabella), che mostra l'output delle query in formato tabulare.

Per ottenere il piano di esecuzione delle query, esegui `EXPLAIN ANALYZE` o `EXPLAIN ANALYZE VERBOSE` per ottenere ulteriori informazioni sulle prestazioni delle query. Per ulteriori informazioni, consulta Leggere i piani di Aurora DSQL EXPLAIN.

> **Tip**
>
> Il `EXPLAIN ANALYZE VERBOSE` comando visualizza le stime sull'utilizzo della DPU, inclusi i valori Compute, Read, Write e Total DPU, fornendo una visibilità immediata delle risorse utilizzate dalle singole istruzioni SQL.

Editor di query: utilizzo JupyterLab con Aurora DSQL

Questa guida fornisce step-by-step istruzioni su come connettersi e interrogare Amazon Aurora DSQL utilizzando Python. JupyterLab JupyterLab è un popolare ambiente informatico interattivo che combina codice, testo e visualizzazioni in un unico documento. È ampiamente utilizzato per la scienza dei dati e le applicazioni di ricerca.

Le istruzioni seguenti illustreranno le basi dell'utilizzo di Aurora DSQL sia nell'installazione locale che nell'utilizzo di SageMaker Amazon AI, un servizio JupyterLab di machine learning completamente gestito che fornisce un ambiente ospitato con un'interfaccia utente per i flussi di lavoro di dati.

Nozioni di base

Requisiti

- Un cluster Aurora DSQL
- Credenziali AWS configurate (solo installazione locale)
- Python versione 3.9 o successiva (solo installazione locale)

Utilizzo locale JupyterLab

Per iniziare JupyterLab, gli utenti devono prima installare l'applicazione usando il pip di Python:

```
pip install jupyterlab
```

JupyterLab può quindi essere aperto eseguendo. **`jupyter lab`** Questo aprirà l' JupyterLabapplicazione su localhost:8888, accessibile in un browser. Assicurati di avere le credenziali AWS configurate nel tuo ambiente locale prima di procedere.

Utilizzo di Amazon SageMaker AI

Nella console AWS, vai alla pagina della console Amazon SageMaker AI e quindi alla sezione Notebook in Applicazioni e. IDEs Da lì puoi selezionare Crea un'istanza di notebook per iniziare a creare un ambiente. SageMaker Seleziona un tipo di istanza e una piattaforma prima di fare clic su Crea istanza notebook.

Consulta la documentazione sulla configurazione di Amazon SageMaker AI per ulteriori informazioni sulle opzioni di configurazione e istanza.

> **Note**
>
> Avviso: l'utilizzo di Amazon SageMaker AI può comportare addebiti sul tuo account AWS.

Una volta che l' SageMaker istanza diventa attiva, puoi aprirla dalla sezione Notebook instances con Apri JupyterLab. Prima di iniziare a utilizzare Aurora DSQL sul notebook, è necessario fornire l'accesso al cluster DSQL nel ruolo IAM dell' SageMaker istanza. Il modo più semplice per farlo è seguire il link al ruolo IAM nella pagina dell'istanza del notebook. Da lì puoi modificare le policy allegate al tuo ruolo SageMaker IAM. Vedi Autenticazione e autorizzazione per ulteriori informazioni sulla configurazione di una policy IAM per consentire l'accesso ad Aurora DSQL.

Connessione ad Aurora DSQL tramite JupyterLab

Dopo aver configurato un' JupyterLab istanza, i passaggi per connettersi ad Aurora DSQL sono gli stessi a livello locale e in AI. SageMaker Crea un taccuino Python 3 vuoto, in cui puoi aggiungere celle con codice Python.

In una cella Python, scarica il certificato root di Amazon dal trust store ufficiale:

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Per connetterti ad Aurora DSQL, installa prima il connettore Aurora DSQL per Python e il driver Psycopg in una cella Python, quindi importalo:

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Con il connettore importato, puoi quindi creare una configurazione DSQL e connetterti. Il connettore Python Aurora DSQL gestirà automaticamente la creazione di un token di autenticazione su ogni connessione.

```
config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin"
    }

conn = dsql.connect(**config)
```

Dopo aver eseguito il codice, ora dovresti avere una connessione Psycopg ad Aurora DSQL. È quindi possibile eseguire query utilizzando il cursore Psycopg e fornendo la query SQL. Consulta la documentazione di Psycopg per ulteriori informazioni sull'uso di Psycopg con un database compatibile con Postgres. Questa query risulterà in un elenco di tuple in. `results_list`

```
with conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

È quindi possibile utilizzare framework Python come Pandas per analizzare o visualizzare i risultati delle query, ad esempio:

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

Notebook di esempio

Un notebook di esempio che utilizza Aurora DSQL è disponibile nell'archivio degli esempi Aurora DSQL.

Approfondimenti

Documentazione sulla configurazione di Amazon SageMaker AI

Connettore Aurora DSQL per Python

Documentazione Pandas

Backup e ripristino per Amazon Aurora DSQL

Amazon Aurora DSQL permette di soddisfare i requisiti di conformità alle normative e di continuità aziendale attraverso l'integrazione con AWS Backup, un servizio di protezione dei dati completamente gestito che semplifica la centralizzazione e l'automazione dei backup nei servizi AWS, nel cloud e on-premises. Il servizio semplifica la creazione, la gestione e il ripristino dei backup per i cluster Aurora DSQL a Regione singola e multi-Regione.

Le caratteristiche principali comprendono:

- Gestione centralizzata dei backup tramite la Console di gestione AWS, l'SDK o la AWS CLI
- Backup completi del cluster
- Pianificazioni di backup automatizzate e policy di conservazione
- Funzionalità tra regioni e tra account
- Configurazione WORM (write-once, read-many) per tutti i backup archiviati

Per maggiori informazioni sulle funzionalità di AWS Backup Vault Lock e un elenco completo delle funzionalità di AWS Backup disponibili per Aurora DSQL, consultare Vantaggi di Vault Lock e disponibilità delle funzionalità di AWS Backup nella Guida per gli sviluppatori di AWS Backup.

Nozioni di base su AWS Backup

AWS Backup crea copie complete dei cluster Aurora DSQL. È possibile iniziare a utilizzare AWS Backup per Aurora DSQL seguendo la procedura descritta in Guida introduttiva a AWS Backup:

1. Creazione di backup on-demand per una protezione immediata.
2. Definizione di piani di backup per backup automatizzati e pianificati.
3. Configurazione dei periodi di conservazione e della copia tra regioni.
4. Configurazione del monitoraggio e delle notifiche per le attività di backup.

Ripristino dei backup

Quando si ripristinano i cluster Aurora DSQL, AWS Backup crea sempre nuovi cluster per preservare i dati di origine.

Ripristino dei cluster basati su una Regione singola

Per ripristinare un cluster Aurora DSQL a Regione singola, utilizzare la console: https://console.aws.amazon.com/backup o la CLI per selezionare il punto di ripristino (backup) che si desidera ripristinare. Configurare le impostazioni del nuovo cluster che sarà creato dal backup. Per istruzioni dettagliate, consultare Ripristino di un cluster Aurora DSQL a Regione singola.

Ripristino di cluster multi-Regione

Il ripristino di un cluster Aurora DSQL multi-Regione è supportato sia dalla console: https://console.aws.amazon.com/backup che dalla AWS CLI. Per istruzioni dettagliate, consultare Ripristino di un cluster Aurora DSQL multi-Regione.

Per eseguire il ripristino in un cluster Aurora DSQL multi-Regione, è possibile utilizzare un backup eseguito in un'unica Regione AWS. Tuttavia, prima di iniziare il processo di ripristino, è necessario assicurarsi che esista una copia identica del backup in tutte le Regioni AWS del cluster multi-Regione. Se non si dispone ancora di tali copie, è prima necessario copiare il backup in un'altra Regione AWS che supporti i cluster multi-Regione.

Per abilitare opzioni robuste di disaster recovery e soddisfare i requisiti di conformità, consigliamo di creare copie di backup nelle Regioni AWS chiave. Per la visualizzazione delle Regioni AWS disponibili per Aurora DSQL consultare the section called "Regione AWS disponibilità".

Per istruzioni dettagliate su questi passaggi, consultare la documentazione sul ripristino di Amazon Aurora DSQL.

Monitoraggio e conformità

AWS Backup offre una visibilità completa sulle operazioni di backup e ripristino con le seguenti risorse.

- Una dashboard centralizzata per il monitoraggio dei processi di backup e ripristino
- Integrazione con CloudWatch e CloudTrail.
- AWS Backup Audit Manager per la rendicontazione e il controllo della conformità.

Consultare Registrazione dei log delle operazioni di Aurora DSQL utilizzando AWS CloudTrail per ulteriori informazioni sulla registrazione dei record delle operazioni eseguite da un utente, un ruolo o un Servizio AWS durante l'utilizzo di Aurora DSQL.

Risorse aggiuntive

Per maggiori informazioni sulle funzionalità di AWS Backup e sul suo utilizzo in combinazione con Aurora DSQL, consultare le seguenti risorse:

- Policy gestite per AWS Backup
- Ripristino di Amazon Aurora DSQL
- Servizi supportati da Regione AWS
- Crittografia per i backup in AWS Backup

Utilizzando AWS Backup per Aurora DSQL, si implementa una strategia di backup robusta, conforme e automatizzata che protegge le risorse critiche del database riducendo al minimo il sovraccarico amministrativo. Indipendentemente dal fatto che si gestisca un singolo cluster o un'implementazione complessa in più regioni, AWS Backup fornisce gli strumenti necessari per garantire che i dati rimangano sicuri e recuperabili.

Monitoraggio e registrazione dei log per Aurora DSQL

Il monitoraggio e la registrazione dei log sono importanti per garantire l'affidabilità, la disponibilità e le prestazioni delle risorse di Amazon Aurora DSQL. È necessario monitorare e raccogliere i dati di log da tutte le componenti delle risorse di Aurora DSQL per semplificare il debug di eventuali malfunzionamenti distribuiti.

- Amazon CloudWatch monitora AWS le tue risorse e le applicazioni su cui esegui AWS in tempo reale. È possibile raccogliere e tenere traccia dei parametri, creare pannelli di controllo personalizzati e impostare allarmi per inviare una notifica o intraprendere azioni quando un parametro specificato raggiunge una determinata soglia. Ad esempio, puoi tenere CloudWatch traccia dell'utilizzo della CPU o di altri parametri delle tue EC2 istanze Amazon e avviare automaticamente nuove istanze quando necessario. Per ulteriori informazioni, consulta la Amazon CloudWatch User Guide.
- AWS CloudTrailacquisisce le chiamate API e gli eventi correlati effettuati da o per conto tuo Account AWS e invia i file di log a un bucket Amazon S3 da te specificato. Puoi identificare quali utenti e account hanno chiamato AWS, l'indirizzo IP di origine da cui sono state effettuate le chiamate e quando sono avvenute le chiamate. Per ulteriori informazioni, consulta la Guida per l'utente AWS CloudTrail.

Monitoraggio di Aurora DSQL con Amazon CloudWatch

È possibile monitorare Aurora DSQL utilizzando CloudWatch, che raccoglie i dati non elaborati e li elabora trasformandoli in parametri leggibili quasi in tempo reale. CloudWatch conserva queste statistiche per 15 mesi, permettendo di ottenere una prospettiva migliore sulle prestazioni delle applicazioni o dei servizi web. È possibile impostare allarmi che controllano soglie specifiche e inviare notifiche o intraprendere azioni quando queste soglie vengono raggiunte. Esaminare le seguenti metriche di utilizzo e osservabilità disponibili per Aurora DSQL.

Per maggiori informazioni, consultare la Guida per l'utente di Amazon CloudWatch.

Osservabilità e prestazioni

Questa tabella descrive le metriche di osservabilità per Aurora DSQL. Include metriche per il monitoraggio delle transazioni di sola lettura e totali per fornire la caratterizzazione complessiva del carico di lavoro. Sono incluse metriche utilizzabili come i timeout delle query e il tasso di conflitto

OCC per aiutare a identificare problemi di prestazioni e conflitti di concorrenza. Le metriche relative alla sessione, sia attive che totali, offrono informazioni sul carico attuale del sistema.

Nome della metrica CloudWatch	Parametro	Unità	Descrizione
ReadOnlyTransactions	Read-only transactions	none	The number of read-only transactions
TotalTransactions	Total transactions	none	The total number of transactions executed on the system, including read-only transactions.
QueryTimeouts	Query timeouts	none	The number of queries which have timed out due to hitting the maximum transaction time
OccConflicts	OCC conflicts	none	The number of transactions aborted due to key level OCC
CommitLatency	Commit Latency	milliseconds	Time spent by commit phase of query execution (P50)
BytesWritten	Bytes Written	bytes	Bytes written to storage
BytesRead	Bytes Read	bytes	Bytes read from storage
ComputeTime	QP compute time	milliseconds	QP wall clock time
ClusterStorageSize	Cluster Storage Size	bytes	Cluster size

Parametri di utilizzo

Aurora DSQL misura tutte le attività basate sulle richieste, come l'elaborazione delle query, le letture e le scritture, utilizzando un'unica unità di fatturazione normalizzata denominata Distributed Processing Unit (DPU).

Nome della metrica CloudWatch	Parametro	Dimensione: ResourceId	Unità	Descrizione
WriteDPU	Write Units	<cluster-id>	DPU	Approximates the write active-use component of your Aurora DSQL cluster DPU usage.
MultiRegionWriteDPU	Multi-Region Write Units	<cluster-id>	DPU	Applicable for Multi-Region clusters: Approximates the multi-Region write active-use component of your Aurora DSQL cluster DPU usage.
ReadDPU	Read Units	<cluster-id>	DPU	Approximates the read active-use component of your Aurora DSQL cluster DPU usage.
ComputeDPU	Compute Units	<cluster-id>	DPU	Approximates the compute active-use

Nome della metrica CloudWatch	Parametro	Dimensione: ResourceId	Unità	Descrizione
				component of your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

Registrazione dei log delle operazioni di Aurora DSQL utilizzando AWS CloudTrail

Amazon Aurora DSQL è integrato con AWS CloudTrail, un servizio che fornisce un record delle azioni intraprese da un utente, un ruolo o un Servizio AWS. In CloudTrail sono disponibili due tipi di eventi: eventi di gestione ed eventi di dati. Gli eventi di gestione vengono emessi per controllare le modifiche alla configurazione delle risorse AWS. Gli eventi di dati registrano l'utilizzo delle risorse di AWS, tipicamente sul piano dati del servizio.

CloudTrail acquisisce tutte le chiamate API verso Aurora DSQL come eventi. Aurora DSQL registra l'attività della console come eventi di gestione. Inoltre, acquisisce i tentativi di connessione autenticati ai cluster come eventi di dati.

Le informazioni raccolte da CloudTrail consentono di determinare la richiesta effettuata ad Aurora DSQL, l'indirizzo IP da cui è partita la richiesta, il momento in cui è stata eseguita, l'identità dell'utente autore della richiesta e altri dettagli.

CloudTrail è attivo per impostazione predefinita nell'Account AWS quando questo viene creato e l'utente dispone dell'accesso alla cronologia degli eventi di CloudTrail. La cronologia degli eventi di CloudTrail fornisce una registrazione visualizzabile, ricercabile, scaricabile e immutabile degli eventi di gestione verificatisi negli ultimi 90 giorni in una Regione AWS. Per maggiori informazioni,

consultare Lavorare con la cronologia degli eventi di CloudTrail nella Guida per l'utente di AWS CloudTrail. CloudTrail non prevede costi per la registrazione della cronologia degli eventi.

Per creare una registrazione continua degli eventi nel proprio account AWS, inclusi gli eventi relativi ad Aurora DSQL, creare un trail o un archivio di dati di eventi AWS CloudTrail Lake (una soluzione centralizzata di archiviazione e analisi per gli eventi AWS CloudTrail). Per maggiori informazioni sulla creazione di trail, consultare Lavorare con i trail di CloudTrail. Per maggiori informazioni sulla configurazione e la gestione degli archivi di dati di eventi, consultare archivi di dati di eventi CloudTrail Lake.

Eventi di gestione di Aurora DSQL in CloudTrail

Gli eventi di gestione di CloudTrail forniscono informazioni sulle operazioni di gestione eseguite sulle risorse nell'account AWS. Queste operazioni sono definite anche operazioni del piano di controllo (control-plane). Per impostazione predefinita, CloudTrail acquisisce gli eventi di gestione nella cronologia degli eventi.

Amazon Aurora DSQL registra i log di tutte le operazioni sul piano di controllo (control-plane) come eventi di gestione. Per un elenco delle operazioni sul piano di controllo (control-plane) di Amazon Aurora DSQL che Aurora DSQL registra in CloudTrail, consulta la Guida di riferimento alle API di Aurora DSQL.

Registrazione dei log del piano di controllo (control-plane)

Amazon Aurora DSQL registra i log delle seguenti operazioni sul piano di controllo (control-plane) di Aurora DSQL su CloudTrail come eventi di gestione.

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName
- ListClusters
- ListTagsForResource
- TagResource
- UntagResource
- UpdateCluster

Backup e ripristino dei log

Amazon Aurora DSQL registra i log delle seguenti operazioni di backup e ripristino di Aurora DSQL su CloudTrail come eventi di gestione.

- `StartBackupJob`
- `StopBackupJob`
- `GetBackupJob`
- `StartRestoreJob`
- `StopRestoreJob`
- `GetRestoreJob`

Per maggiori informazioni sulla protezione dei cluster Aurora DSQL con AWS Backup, consultare Backup e ripristino per Amazon Aurora DSQL.

Log di AWS KMS

Amazon Aurora DSQL registra i log delle seguenti operazioni di AWS KMS su CloudTrail come eventi di gestione.

- `GenerateDataKey`
- `Decrypt`

Per maggiori informazioni su come i log di CloudTrail tengono traccia delle richieste inviate da Aurora DSQL a AWS KMS per conto dell'utente, consultare Monitoraggio dell'interazione di Aurora DSQL con AWS KMS.

Eventi di dati di Aurora DSQL in CloudTrail

Gli eventi di dati di CloudTrail forniscono tipicamente informazioni sulle operazioni eseguite su una risorsa o al suo interno. Questi vengono utilizzati anche per acquisire le operazioni sul piano dati del servizio. Gli eventi di dati sono spesso attività che interessano volumi elevati di dati. Per impostazione predefinita, CloudTrail non registra gli eventi di dati. La cronologia degli eventi di CloudTrail non registra gli eventi di dati.

Per maggiori informazioni su come registrare i log degli eventi di dati, consultare Registrazione di eventi di dati con Console di gestione AWS e Registrazione di eventi di dati con AWS Command Line Interface nella Guida per l'utente di AWS CloudTrail.

Per gli eventi di dati sono previsti costi aggiuntivi. Per maggiori informazioni sui prezzi di CloudTrail, consultare Prezzi di AWS CloudTrail.

Per Aurora DSQL, CloudTrail acquisisce qualsiasi tentativo di connessione effettuato a un cluster Aurora DSQL come evento di dati. La tabella seguente elenca i tipi di risorse di Aurora DSQL per i quali è possibile registrare i log degli eventi di dati. La colonna Tipo di risorsa (console) mostra il valore da scelto dall'elenco Tipo di risorsa nella console di CloudTrail. La colonna Valore resources.type mostra il valore `resources.type`, da specificare quando si configurano selettori di eventi avanzati utilizzando la AWS CLI o le API di CloudTrail. La colonna API sui dati registrate su CloudTrail mostra le chiamate API registrate su CloudTrail per il tipo di risorsa.

Tipo di risorsa (console)	Valore resources.type	API sui dati registrate in CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

È possibile configurare selettori di eventi avanzati per filtrare i campi `eventName` e `resources.ARN` per registrare solo i log degli eventi filtrati. Per maggiori informazioni su questi campi, consultare AdvancedFieldSelector nella Guida di riferimento delle API di AWS CloudTrail.

L'esempio seguente mostra come utilizzare la AWS CLI per configurare `dsql-data-events-trail` affinché riceva eventi di dati per Aurora DSQL.

```
aws cloudtrail put-event-selectors \
--region us-east-1 \
--trail-name dsql-data-events-trail \
--advanced-event-selectors '[{
"Name": "Log DSQL Data Events",
    "FieldSelectors": [
       { "Field": "eventCategory", "Equals": ["Data"] },
       { "Field": "resources.type", "Equals": ["AWS::DSQL::Cluster"] } ]}]'
```

Sicurezza in Amazon Aurora DSQL

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di data center e architetture di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra te e te. AWS Il modello di responsabilità condivisa descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi inCloud AWS. AWSfornisce inoltre servizi che è possibile utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWSProgrammi di AWS conformità dei Programmi di conformità dei di . Per informazioni sui programmi di conformità che si applicano ad Amazon Aurora DSQL, consulta AWSServices in Scope by Compliance Program by Compliance Program.
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. L'utente è anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della propria azienda e le leggi e normative vigenti.

Questa documentazione consente di comprendere come applicare il modello di responsabilità condivisa quando si usa Aurora DSQL. I seguenti argomenti illustrano come configurare Aurora DSQL per soddisfare gli obiettivi di sicurezza e conformità. Scopri anche come utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere le tue risorse Aurora DSQL.

Argomenti

- Resilienza in Amazon Aurora DSQL
- Sicurezza dell'infrastruttura in Amazon Aurora DSQL
- Analisi della configurazione e delle vulnerabilità in Amazon Aurora DSQL
- Prevenzione del confused deputy tra servizi
- Best practice di sicurezza di Aurora DSQL

AWSpolitiche gestite per Amazon Aurora DSQL

Una policy AWS gestita è una policy autonoma creata e amministrata da. AWS AWSle politiche gestite sono progettate per fornire autorizzazioni per molti casi d'uso comuni, in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Tieni presente che le policy AWS gestite potrebbero non concedere le autorizzazioni con il privilegio minimo per i tuoi casi d'uso specifici, poiché sono disponibili per tutti i clienti. AWS Si consiglia pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i propri casi d'uso.

Non è possibile modificare le autorizzazioni definite nelle politiche gestite. AWS Se AWS aggiorna le autorizzazioni definite in una politica AWS gestita, l'aggiornamento ha effetto su tutte le identità principali (utenti, gruppi e ruoli) a cui è associata la politica. AWSè più probabile che aggiorni una policy AWS gestita quando ne Servizio AWS viene lanciata una nuova o quando diventano disponibili nuove operazioni API per i servizi esistenti.

Per ulteriori informazioni, consultare Policy gestite da AWS nella Guida per l'utente di IAM.

AWSpolitica gestita: AmazonAurora DSQLFull accesso

È possibile associare la policy `AmazonAuroraDSQLFullAccess` a utenti, gruppi e ruoli.

Questa policy concede autorizzazioni che forniscono l'accesso completo come amministratore ad Aurora DSQL. Le entità principali con queste autorizzazioni possono:

- Creare, eliminare e aggiornare i cluster Aurora DSQL, inclusi i cluster multi-Regione
- Gestisci le politiche in linea del cluster (creazione, visualizzazione, aggiornamento ed eliminazione delle politiche)

- Aggiungere e rimuover tag dai cluster
- Elencare i cluster e visualizzare le informazioni sui singoli cluster
- Visualizzare i tag associati ai cluster Aurora DSQL
- Collegarsi al database come qualsiasi utente, incluso l'amministratore
- Eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio dei processi di backup e ripristino
- Utilizza AWS KMS chiavi gestite dal cliente per la crittografia dei cluster
- Visualizza tutte le metriche del loro account CloudWatch
- Usa AWS Fault Injection Service (AWS FIS) per inserire errori nei cluster Aurora DSQL per i test di tolleranza ai guasti
- Creare ruoli collegati al servizio per il servizio `dsql.amazonaws.com`, necessari per la creazione dei cluster

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `dsql` - Concede alle entità principali l'accesso completo ad Aurora DSQL.
- `cloudwatch`—concede l'autorizzazione a pubblicare punti dati metrici su Amazon. CloudWatch
- `iam` - Concede le autorizzazioni per creare un ruolo collegato al servizio.
- `backup and restore` - Concede le autorizzazioni per avviare, interrompere e monitorare i processi di backup e ripristino per i cluster Aurora DSQL.
- `kms` - Concede le autorizzazioni necessarie per convalidare l'accesso alle chiavi gestite dal cliente utilizzate per la crittografia dei cluster Aurora DSQL durante la creazione, l'aggiornamento o la connessione ai cluster.
- `fis`—concede le autorizzazioni di utilizzo (AWS Fault Injection ServiceAWS FIS) per inserire errori nei cluster Aurora DSQL per i test di tolleranza agli errori.

È possibile trovare la policy `AmazonAuroraDSQLFullAccess` nella console IAM e nella Guida di riferimento alle policy gestite di AWS.

AWSpolitica gestita: AmazonAurora DSQLRead OnlyAccess

È possibile associare la policy `AmazonAuroraDSQLReadOnlyAccess` a utenti, gruppi e ruoli.

Consente l'accesso in lettura ad Aurora DSQL. Le entità principali con queste autorizzazioni possono elencare i cluster e visualizzare informazioni sui singoli cluster. Possono vedere i tag allegati ai cluster Aurora DSQL e visualizzare le policy in linea del cluster. Possono recuperare e visualizzare qualsiasi metrica del tuo account. CloudWatch

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `dsql` - Concede autorizzazioni di sola lettura su tutte le risorse in Aurora DSQL.
- `cloudwatch`— concede l'autorizzazione a recuperare quantità batch di dati metrici ed eseguire CloudWatch calcoli metrici sui dati recuperati

È possibile trovare la policy `AmazonAuroraDSQLReadOnlyAccess` nella console IAM e nella Guida di riferimento alle policy gestite di AWS.

AWSpolitica gestita: AmazonAurora DSQLConsole FullAccess

È possibile associare la policy `AmazonAuroraDSQLConsoleFullAccess` a utenti, gruppi e ruoli.

Consente l'accesso amministrativo completo ad Amazon Aurora DSQL tramite la Console di gestione AWS. Le entità principali con queste autorizzazioni possono:

- Creare, eliminare e aggiornare i cluster Aurora DSQL, inclusi i cluster multi-Regione, con la console
- Gestisci le politiche in linea del cluster tramite la console (creazione, visualizzazione, aggiornamento ed eliminazione delle politiche)
- Elencare i cluster e visualizzare le informazioni sui singoli cluster
- Visualizzare i tag su qualsiasi risorsa dell'account
- Collegarsi al database come qualsiasi utente, incluso l'amministratore
- Eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio dei processi di backup e ripristino
- Utilizza AWS KMS chiavi gestite dal cliente per la crittografia dei cluster
- Avvia AWS CloudShell da Console di gestione AWS

- Visualizza tutte le metriche dal CloudWatch tuo account
- Usa AWS Fault Injection Service (AWS FIS) per inserire errori nei cluster Aurora DSQL per i test di tolleranza ai guasti
- Creare ruoli collegati al servizio per il servizio `dsql.amazonaws.com`, necessari per la creazione dei cluster

Puoi trovare la `AmazonAuroraDSQLConsoleFullAccess` policy sulla console IAM e AmazonAuroraDSQLConsoleFullAccessnella Managed Policy Reference Guide. AWS

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `dsql` - Concede autorizzazioni amministrative complete a tutte le risorse in Aurora DSQL tramite la Console di gestione AWS.
- `cloudwatch`—concede l'autorizzazione a recuperare quantità in batch di dati metrici ed eseguire CloudWatch calcoli metrici sui dati recuperati.
- `tag`—concede l'autorizzazione a restituire le chiavi e i valori dei tag attualmente in uso nell'account specificato per la chiamata. Regione AWS
- `backup and restore` - Concede le autorizzazioni per avviare, interrompere e monitorare i processi di backup e ripristino per i cluster Aurora DSQL.
- `kms` - Concede le autorizzazioni necessarie per convalidare l'accesso alle chiavi gestite dal cliente utilizzate per la crittografia dei cluster Aurora DSQL durante la creazione, l'aggiornamento o la connessione ai cluster.
- `cloudshell`—concede le autorizzazioni all'avvio AWS CloudShell per interagire con Aurora DSQL.
- `ec2` - Concede l'autorizzazione a visualizzare le informazioni sugli endpoint Amazon VPC necessarie per le connessioni Aurora DSQL.
- `fis`—concede le autorizzazioni da utilizzare AWS FIS per inserire errori nei cluster Aurora DSQL per il test della tolleranza agli errori.
- `access-analyzer:ValidatePolicy`concede l'autorizzazione per il linter nell'editor delle politiche, che fornisce feedback in tempo reale su errori, avvisi e problemi di sicurezza nella politica corrente.

- `fis`—concede le autorizzazioni di utilizzo (AWS Fault Injection ServiceAWS FIS) per inserire errori nei cluster Aurora DSQL per i test di tolleranza agli errori.

È possibile trovare la policy `AmazonAuroraDSQLConsoleFullAccess` nella console IAM e nella Guida di riferimento alle policy gestite di AWS.

AWSpolitica gestita: Aurora DSQLService RolePolicy

Non puoi collegare Aurora DSQLService RolePolicy alle tue entità IAM. Questa policy è allegata a un ruolo collegato al servizio che consente ad Aurora DSQL di accedere alle risorse dell'account.

Puoi trovare la `AuroraDSQLServiceRolePolicy` policy sulla console IAM e Aurora DSQLService RolePolicy nella AWS Managed Policy Reference Guide.

Aurora DSQL si aggiorna alle policy gestite AWS

Visualizza i dettagli sugli aggiornamenti delle politiche AWS gestite per Aurora DSQL da quando questo servizio ha iniziato a tenere traccia di queste modifiche. Per gli avvisi automatici sulle modifiche apportate a questa pagina, sottoscrivi il feed RSS nella pagina della cronologia dei documenti di Aurora DSQL.

Modifica	Descrizione	Data
AmazonAuroraDSQLFullAccesso e aggiornamento AmazonAurora DSQLConsole FullAccess	È stato aggiunto il supporto per l'integrazione AWS Fault Injection Service (AWS FIS) con Aurora DSQL. Ciò consente di eseguire l'iniezione di guasti in cluster Aurora DSQL a Regione singola e multi-Regione per testare la tolleranza ai guasti delle applicazioni. È possibile creare modelli di esperimento nella AWS FIS console	19 agosto 2025

Modifica	Descrizione	Data
	per definire scenari di errore e indirizzare cluster Aurora DSQL specifici per i test. Per ulteriori informazioni su queste politiche, consulta AmazonAurora DSQLFull Access and. AmazonAur oraDSQLConsoleFullAccess	
AmazonAuroraDSQLFu llAccesso AmazonAurora DSQLRead OnlyAccess e AmazonAurora DSQLConsole FullAccess aggiornamento	È stato aggiunto il supporto per le policy basate sulle risorse (RBP) con nuove autorizzazioni:, e. `PutCluste rPolicy GetCluste rPolicy DeleteClu sterPolicy` Queste autorizzazioni consentono di gestire le policy in linea collegate ai cluster Aurora DSQL per un controllo granulare degli accessi. Per ulteriori informazioni, vedere Access, e. AmazonAur ora DSQLFull AmazonAur oraDSQLReadOnlyAcc essAmazonAuroraDSQLCo nsoleFullAccess	15 ottobre 2025

Modifica	Descrizione	Data
AmazonAuroraDSQLFullAggiornamento dell'accesso	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio dei processi. Aggiunge inoltre la possibilità di utilizzare chiavi KMS gestite dal cliente per la crittografia dei cluster. Per ulteriori informazioni, vedere AmazonAuroraDSQLFullAccesso e utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	21 maggio 2025
AmazonAuroraDSQLConsoleFullAccess update	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL tramite la AWS Console Home. Ciò include l'avvio, l'arresto e il monitoraggio dei processi. Supporta anche l'utilizzo di chiavi KMS gestite dal cliente per la crittografia dei cluster e l'avvio di AWS CloudShell. Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	21 maggio 2025

Modifica	Descrizione	Data
AmazonAuroraDSQLFullAccedi all'aggiornamento	La politica aggiunge quattro nuove autorizzazioni per creare e gestire cluster di database su più livelliRegioni AWS:`PutMultiRegionProperties` , `PutWitnessRegion` `AddPeerCluster` , e. `RemovePeerCluster` Queste autorizzazioni includono controlli a livello di risorsa e chiavi di condizione in modo da poter controllare quali cluster possono essere modificati dagli utenti. La policy aggiunge anche l'autorizzazione `GetVpcEndpointServiceName` per consentire di connettersi ai cluster Aurora DSQL tramite AWS PrivateLink. Per ulteriori informazioni, vedere Per ulteriori informazioni, vedere AmazonAuroraDSQLFullAccesso e utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

Modifica	Descrizione	Data
AmazonAuroraDSQLReadOnlyAccess update	Include la possibilità di determinare il nome corretto del servizio endpoint VPC durante la connessione ai cluster Aurora DSQL tramite AWS PrivateLink Aurora DSQL crea endpoint unici per cella, quindi questa API aiuta a identificare l'endpoint corretto per il cluster ed evitare errori di connessione. Per ulteriori informazioni, vedere AmazonAuroraDSQLReadOnlyAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

Modifica	Descrizione	Data
AmazonAuroraDSQLConsoleFullAccess update	Aggiunge nuove autorizzazioni ad Aurora DSQL per supportare la gestione di cluster multi-Regione e la connessione agli endpoint VPC. Le nuove autorizzazioni includono: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` , `RemovePeerCluster` e `GetVpcEndpointServiceName` Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025
AuroraDsqlServiceLinkedRolePolicy update	Aggiunge la possibilità di pubblicare metriche nei namespace `AWS/AuroraDSQL` e `AWS/Usage` CloudWatch della policy. Ciò consente al servizio o al ruolo associato di inviare dati più completi sull'utilizzo e sulle prestazioni nell'ambiente. CloudWatch Per ulteriori informazioni, vedere AuroraDsqlServiceLinkedRolePolicye Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	8 maggio 2025

Modifica	Descrizione	Data
Pagina creata	Ha iniziato a tracciare le policy AWS gestite relative ad Amazon Aurora DSQL	3 dicembre 2024

Protezione dei dati in Amazon Aurora DSQL

Alla protezione dei dati si applica il modello di responsabilità condivisa. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che esegue tutto il Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. Il cliente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i servizi utilizzati. Per maggiori informazioni sulla privacy dei dati, consulta le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al Modello di responsabilità condivisa e GDPR nel Blog sulla sicurezza.

Ai fini della protezione dei dati, ti consigliamo di proteggere le credenziali e configurare i singoli utenti con o. AWS IAM Identity Center AWS Identity and Access Management In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti conAWS CloudTrail. Per informazioni sull'utilizzo dei trail per acquisire le attività, consulta Utilizzo dei percorsi CloudTrail nella Guida per l'utente di CloudTrail.
- Utilizza le soluzioni di crittografia, insieme a tutti i controlli di sicurezza di default all'interno dei Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.

Consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori o utilizzi la console, l'API o AWSSDKs. AWS CLI I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di

diagnostica. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Crittografia dei dati

Amazon Aurora DSQL offre un'infrastruttura di storage estremamente durevole, concepita per lo archiviazione di dati mission-critical e primari. I dati sono archiviati in modo ridondante su più dispositivi, in più strutture di una Regione Aurora DSQL.

Crittografia dei dati in transito

Per impostazione predefinita, è configurata la crittografia in transito. Aurora DSQL utilizza TLS per crittografare tutto il traffico tra il client SQL e Aurora DSQL.

Crittografia e firma dei dati in transito tra AWS CLI client SDK o API e endpoint Aurora DSQL:

- Aurora DSQL fornisce endpoint HTTPS per la crittografia dei dati in transito.
- Per proteggere l'integrità delle richieste API ad Aurora DSQL, le chiamate API devono essere firmate dal chiamante. Le chiamate sono firmate da un certificato X.509 o dalla chiave di accesso AWS segreta del cliente in base al processo di firma della versione 4 di firma (Sigv4). Per maggiori informazioni, consulta Processo di firma Signature Version 4 nella Riferimenti generali di AWS.
- Usa AWS CLI o uno dei due per effettuare richieste AWS SDKs a. AWS Questi strumenti firmano automaticamente le richieste con la chiave di accesso specificata al momento della configurazione.

Conformità a FIPS

Gli endpoint del piano dati Aurora DSQL (endpoint del cluster utilizzati per le connessioni al database) utilizzano moduli crittografici convalidati FIPS 140-2 per impostazione predefinita. Non sono necessari endpoint FIPS separati per le connessioni al cluster.

Per le operazioni sul piano di controllo, Aurora DSQL fornisce endpoint FIPS dedicati nelle regioni supportate. Per ulteriori informazioni sugli endpoint FIPS del piano di controllo, vedere Endpoint e quote Aurora DSQL in. Riferimenti generali di AWS

Per la crittografia a riposo, consulta Crittografia dei dati a riposo in Aurora DSQL.

Riservatezza del traffico inter-rete

Le connessioni sono protette sia tra Aurora DSQL e le applicazioni locali sia tra Aurora DSQL e altre risorse all'interno delle stesse. AWS Regione AWS

Sono disponibili due opzioni di connettività tra la rete privata e: AWS

- Una connessione AWS Site-to-Site VPN. Per maggiori informazioni, consulta Che cos'è AWS Site-to-Site VPN?
- Una Direct Connect connessione. Per ulteriori informazioni, vedi Cos'èDirect Connect?

È possibile ottenere l'accesso ad Aurora DSQL tramite la rete utilizzando le operazioni API pubblicate da AWS. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Protezione dei dati nelle Regioni testimone

Quando si crea un cluster multi-Regione, una Regione testimone consente il ripristino automatico degli errori partecipando alla replica sincrona delle transazioni crittografate. Se un cluster in peering diventa non disponibile, la Regione testimone rimane disponibile per convalidare ed elaborare le scritture sul database, garantendo l'assenza di perdita di disponibilità.

Le Regioni testimone proteggono e mantengono al sicuro i dati tramite queste funzionalità imposte come requisito di progettazione:

- La Regione testimone riceve e archivia i registri dei log delle sole transazioni crittografate. Non ospita, archivia o trasmette mai le chiavi di crittografia.
- La Regione testimone si concentra esclusivamente sulla registrazione delle transazioni di scrittura e sulle funzioni di quorum. Per requisito di progettazione non è in grado di leggere i dati.
- La Regione testimone funziona senza endpoint di connessione al cluster o elaboratori di query. Ciò impedisce l'accesso al database da parte degli utenti.

Per maggiori informazioni sulle Regioni testimoni, consulta Configurazione di cluster multi-Regione.

Configurazione dei SSL/TLS certificati per le connessioni Aurora DSQL

Aurora DSQL richiede che tutte le connessioni utilizzino la crittografia Transport Layer Security (TLS). Per stabilire connessioni sicure, il sistema client deve affidarsi all'Amazon Root Certificate Authority

(Amazon Root CA 1). Questo certificato è preinstallato su molti sistemi operativi. Questa sezione fornisce istruzioni per verificare il certificato Amazon Root CA 1 preinstallato su vari sistemi operativi e guida l'utente attraverso il processo di installazione manuale del certificato, qualora non fosse già presente.

Si consiglia di utilizzare PostgreSQL versione 17.

> **Important**
>
> Per gli ambienti di produzione, si consiglia di utilizzare la modalità SSL `verify-full` per garantire il massimo livello di sicurezza della connessione. Questa modalità verifica che il certificato del server sia firmato da un'autorità di certificazione affidabile e che il nome host del server corrisponda al certificato.

Verifica dei certificati preinstallati

Nella maggior parte dei sistemi operativi, Amazon Root CA 1 è già preinstallato. Per convalidarlo, è possibile completare la procedura seguente.

Linux () RedHat/CentOS/Fedora

Esegui il comando seguente nel terminale:

```
trust list | grep "Amazon Root CA 1"
```

Se il certificato è installato, verrà visualizzato il seguente output:

```
label: Amazon Root CA 1
```

macOS

1. Apri Spotlight Search (Command + Spazio)
2. Cerca Keychain Access
3. Seleziona System Roots in System Keychains
4. Cerca Amazon Root CA 1 nell'elenco dei certificati

Windows

> **Note**
>
> A causa di un problema noto con il client psql per Windows, l'utilizzo dei certificati root di sistema (`sslrootcert=system`) può restituire il seguente errore: `SSL error: unregistered scheme`. È possibile seguire Connessione da Windows come metodo alternativo per connettersi al cluster tramite SSL.

Se Amazon Root CA 1 non è installato nel sistema operativo, procedi nel seguente modo.

Installazione dei certificati

Se il certificato `Amazon Root CA 1` non è preinstallato sul sistema operativo, sarà necessario installarlo manualmente per stabilire connessioni sicure al cluster Aurora DSQL.

Installazione del certificato su Linux

Attieniti alla seguente procedura per installare il certificato Amazon Root CA sui sistemi Linux.

1. Scarica il certificato root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Aggiungi il certificato all'archivio di fiducia:

```
sudo cp ./AmazonRootCA1.pem /etc/pki/ca-trust/source/anchors/
```

3. Aggiorna il CA Trust Store:

```
sudo update-ca-trust
```

4. Verifica l'installazione:

```
trust list | grep "Amazon Root CA 1"
```

Installazione del certificato su macOS

Questi passaggi di installazione dei certificati sono facoltativi. Installazione del certificato su Linux funziona anche per macOS.

1. Scarica il certificato root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Aggiungi il certificato al keychain di sistema:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain
 AmazonRootCA1.pem
```

3. Verifica l'installazione:

```
security find-certificate -a -c "Amazon Root CA 1" -p /Library/Keychains/
System.keychain
```

Connessione con SSL/TLS verifica

Prima di configurare SSL/TLS i certificati per connessioni sicure al cluster Aurora DSQL, assicurati di avere i seguenti prerequisiti.

- PostgreSQL versione 17 installata
- AWS CLIconfigurato con le credenziali appropriate
- Informazioni sugli endpoint del cluster Aurora DSQL

Connessione da Linux

1. Genera e imposta il token di autenticazione:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-
cluster-region --hostname your-cluster-endpoint)
```

2. Connettiti utilizzando certificati di sistema (se preinstallati):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
```

```
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Oppure, connettiti utilizzando un certificato scaricato:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

> **Note**
>
> Per maggiori informazioni sulle impostazioni PGSSLMODE, consulta sslmode nella documentazione sulle Funzioni di controllo della connessione del database di PostgreSQL 17.

Connessione da macOS

1. Genera e imposta il token di autenticazione:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Connettiti utilizzando certificati di sistema (se preinstallati):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Oppure, scarica il certificato root e salvalo con il nome `root.pem` (se il certificato non è preinstallato)

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
```

```
--username admin \
--host your_cluster_endpoint
```

4. Connessione tramite psql:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

Connessione da Windows

Utilizzo del prompt dei comandi

1. Genera il token di autenticazione:

```
aws dsql generate-db-connect-admin-auth-token ^
--region=your-cluster-region ^
--expires-in=3600 ^
--hostname=your-cluster-endpoint
```

2. Imposta la variabile di ambiente della password:

```
set "PGPASSWORD=token-from-above"
```

3. Imposta la configurazione SSL:

```
set PGSSLROOTCERT=C:\full\path\to\root.pem
set PGSSLMODE=verify-full
```

4. Connettiti al database:

```
"C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres ^
--username admin ^
--host your-cluster-endpoint
```

Usando PowerShell

1. Genera e imposta il token di autenticazione:

```
$env:PGPASSWORD = (aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --expires-in=3600 --hostname=your-cluster-endpoint)
```

2. Imposta la configurazione SSL:

```
$env:PGSSLROOTCERT='C:\full\path\to\root.pem'
$env:PGSSLMODE='verify-full'
```

3. Connettiti al database:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
--username admin `
--host your-cluster-endpoint
```

Risorse aggiuntive

- Documentazione di PostgreSQL SSL
- Servizi Amazon Trust

Crittografia dei dati per Amazon Aurora DSQL

Amazon Aurora DSQL crittografa tutti i dati a riposo degli utenti. Per una maggiore sicurezza, questa crittografia utilizza AWS Key Management Service (AWS KMS). Questa funzionalità consente di ridurre gli oneri operativi e la complessità associati alla protezione dei dati sensibili. La crittografia dei dati a riposo permette di:

- Ridurre l'onere operativo legato alla protezione dei dati sensibili
- Creare applicazioni ad alto livello di sicurezza che rispettano rigorosi requisiti normativi e di conformità per la crittografia
- Aggiungere un ulteriore livello di protezione dei dati proteggendo sempre i dati in un cluster crittografato
- Rispettare le policy organizzative, le normative di settore o governative e i requisiti di conformità

La crittografia dei dati a risposo consente di creare applicazioni ad alto livello di sicurezza che rispettano rigorosi requisiti normativi e di conformità per la crittografia. Le sezioni seguenti spiegano

come configurare la crittografia per i database Aurora DSQL nuovi ed esistenti e gestire le chiavi di crittografia.

Argomenti

Tipi di chiave KMS per Aurora DSQL

Aurora DSQL si integra con la gestione delle chiavi di crittografia AWS KMS per i cluster. Per maggiori informazioni sui tipi e gli stati delle chiavi, consulta Concetti di AWS Key Management Service nella Guida per gli sviluppatori di AWS Key Management Service. Quando si crea un nuovo cluster, è possibile selezionare tra i seguenti tipi di chiave KMS per crittografare il cluster:

Chiave di proprietà di AWS

: Tipo di crittografia predefinito. Aurora DSQL possiede la chiave senza costi aggiuntivi per l'utente. Amazon Aurora DSQL decrittografa in modo trasparente i dati del cluster quando si accede a un cluster crittografato. Non è necessario modificare il codice o le applicazioni per utilizzare o gestire i cluster crittografati e tutte le query SQL di Aurora funzionano con i dati crittografati.

Chiave gestita dal cliente

: Tu crei, possiedi e gestisci la chiave del tuo. Account AWS Hai il pieno controllo sulla chiave KMS. AWS KMSsi applicano costi.

La crittografia a riposo utilizzando il Chiave di proprietà di AWS è disponibile senza costi aggiuntivi. Tuttavia, per le chiavi gestite dal cliente vengono AWS KMS addebitati dei costi. Per maggiori informazioni, consulta la pagina Prezzi di AWS KMS.

È possibile passare da un tipo di chiave all'altro in qualsiasi momento. Per maggiori informazioni sui tipi di chiave, consulta Chiavi gestite dal cliente e Chiavi di proprietà di AWS nella Guida per gli sviluppatori di AWS Key Management Service.

> **Note**
>
> La crittografia Aurora DSQL at Rest è disponibile in tutte le regioni in AWS cui è disponibile Aurora DSQL.

Crittografia dei dati a riposo in Aurora DSQL

Amazon Aurora DSQL utilizza Advanced Encryption Standard a 256 bit (AES-256) per crittografare i dati a riposo. Questa crittografia aiuta a proteggere i dati dall'accesso non autorizzato allo storage sottostante. AWS KMSgestisce le chiavi di crittografia per i cluster. È possibile utilizzare l'impostazione predefinita Chiavi di proprietà di AWS o scegliere di utilizzare la propria Chiavi gestite dal cliente AWS KMS. Per maggiori informazioni sulla specificazione e la gestione delle chiavi per i cluster Aurora DSQL, consulta Creazione di un cluster Aurora DSQL crittografato e Rimozione o aggiornamento di una chiave per il cluster Aurora DSQL.

Argomenti

- Chiavi di proprietà di AWS
- Chiavi gestite dal cliente

Chiavi di proprietà di AWS

Aurora DSQL crittografa tutti i cluster per impostazione predefinita con. Chiavi di proprietà di AWS Queste chiavi possono essere utilizzate gratuitamente e ruotano ogni anno per proteggere le risorse degli account. Non è necessario visualizzare, gestire, utilizzare o controllare queste chiavi, quindi non è necessaria alcuna azione per la protezione dei dati. Per ulteriori informazioni in meritoChiavi di proprietà di AWS, consulta la Guida per gli Chiavi di proprietà di AWSsviluppatori. AWS Key Management Service

Chiavi gestite dal cliente

È possibile creare, possedere e gestire chiavi gestite dal cliente in Account AWS. L'utente mantiene il pieno controllo su queste chiavi KMS, comprese le relative policy, il materiale di crittografia, i tag e gli

alias. Per maggiori informazioni sulla gestione delle autorizzazioni, consulta Chiavi gestite dal cliente nella Guida per gli sviluppatori di AWS Key Management Service.

Quando si specifica una chiave gestita dal cliente per la crittografia a livello di cluster, Aurora DSQL crittografa il cluster e tutti i relativi dati regionali con quella chiave. Per prevenire la perdita di dati e mantenere l'accesso al cluster, Aurora DSQL deve accedere alla chiave di crittografia. Se si disabilita la chiave gestita dal cliente, se ne pianifica l'eliminazione o si impone una policy che limita l'accesso al servizio, lo stato di crittografia del cluster diventa `KMS_KEY_INACCESSIBLE`. Quando Aurora DSQL non è in grado di accedere alla chiave, gli utenti non possono connettersi al cluster, lo stato di crittografia del cluster diventa `KMS_KEY_INACCESSIBLE` e il servizio perde l'accesso ai dati del cluster.

Per i cluster multiregionali, i clienti possono configurare la chiave di AWS KMS crittografia di ciascuna regione separatamente e ogni cluster regionale utilizza la propria chiave di crittografia a livello di cluster. Se Aurora DSQL non è in grado di accedere alla chiave di crittografia per un peer in un cluster multi-Regione, lo stato di quel peer diventa `KMS_KEY_INACCESSIBLE` e non è più disponibile per le operazioni di lettura e scrittura. Gli altri peer continuano con la normale operatività.

> **Note**
>
> Se Aurora DSQL non è in grado di accedere alla chiave gestita dal cliente, lo stato di crittografia del cluster diventa `KMS_KEY_INACCESSIBLE`. Dopo aver ripristinato l'accesso alla chiave, il servizio rileverà automaticamente il ripristino entro 15 minuti. Per maggiori informazioni, consulta la sezione Cluster in stato sospeso.
> Per i cluster multi-Regione, se l'accesso alla chiave viene perso per un periodo prolungato, il tempo di ripristino del cluster dipende dalla quantità di dati scritti mentre la chiave era inaccessibile.

Utilizzo AWS KMS e chiavi dati con Aurora DSQL

La funzionalità di crittografia a riposo di Aurora DSQL utilizza una AWS KMS key e una gerarchia di chiavi di dati per proteggere i dati del cluster.

Consigliamo di pianificare la strategia di crittografia prima di implementare il cluster in Aurora DSQL. Se si archiviano dati sensibili o riservati in Aurora DSQL, prendere in considerazione l'inclusione della crittografia lato client nel piano. In questo modo è possibile crittografare i dati il più vicino possibile alla loro origine e garantirne la protezione per tutto il ciclo di vita.

Argomenti

Usare AWS KMS key s con Aurora DSQL

La crittografia dei dati a riposo protegge il cluster Aurora DSQL con una AWS KMS key. Per impostazione predefinita, Aurora DSQL utilizza una Chiave di proprietà di AWS chiave di crittografia multi-tenant creata e gestita in un account del servizio Aurora DSQL. È però possibile crittografare i cluster Aurora DSQL con una chiave gestita dal cliente nel proprio Account AWS. È possibile selezionare una chiave KMS differente per ogni cluster, anche se fa parte di una configurazione multi-Regione.

Quando si crea o si aggiorna il cluster si seleziona la chiave KMS relativa. È possibile modificare la chiave KMS per un cluster in qualsiasi momento, nella console di Aurora DSQL oppure utilizzando l'operazione `UpdateCluster`. Il processo di scambio di chiavi non richiede tempo di inattività e non comporta alcun calo delle prestazioni del servizio.

> **Important**
>
> Aurora DSQL supporta solo chiavi KMS simmetriche. Non è possibile utilizzare una chiave KMS asimmetrica per crittografare i cluster Aurora DSQL.

Una chiave gestita dal cliente fornisce i seguenti vantaggi.

- Il cliente crea e gestisce la chiave KMS, inclusa l'impostazione delle Policy delle chiavi e delle Policy IAM per controllare l'accesso alla chiave KMS. È possibile abilitare e disabilitare la chiave KMS, abilitare e disabilitare la rotazione automatica della chiave ed eliminare la chiave KMS quando non è più in uso.
- È possibile utilizzare una chiave gestita dal cliente con materiale di chiave importato o una chiave gestita dal cliente in un archivio delle chiavi personalizzate di cui il cliente è proprietario e gestore.
- È possibile controllare la crittografia e la decrittografia del cluster Aurora DSQL esaminando le chiamate dell'API Aurora DSQL ai log. AWS KMS AWS CloudTrail

Tuttavia, Chiave di proprietà di AWS è gratuito e il suo utilizzo non influisce sulle quote di risorse o richieste. AWS KMS Le chiavi gestite dal cliente sono soggette a un addebito per ogni chiamata API e a tali chiavi KMS vengono applicate le quote di AWS KMS.

Utilizzo delle chiavi del cluster con Aurora DSQL

Aurora DSQL utilizza for the cluster AWS KMS key per generare e crittografare una chiave dati univoca per il cluster, nota come chiave del cluster.

La chiave del cluster viene utilizzata come chiave di crittografia. Aurora DSQL utilizza questa chiave del cluster per proteggere le chiavi di crittografia dei dati utilizzate per crittografare i dati del cluster. Aurora DSQL genera una chiave di crittografia dei dati univoca per ogni struttura sottostante in un cluster, ma più di un elemento della cluster potrebbe essere protetto dalla stessa chiave di crittografia dei dati.

Per decrittografare la chiave del cluster, Aurora DSQL invia una richiesta a AWS KMS quando si accede per la prima volta a un cluster crittografato. Per mantenere il cluster disponibile, Aurora DSQL verifica periodicamente l'accesso del sistema di decrittazione alla chiave KMS, anche quando non si accede attivamente al cluster.

Aurora DSQL archivia e utilizza la chiave del cluster e le chiavi di crittografia dei dati all'esterno di. AWS KMS Protegge tutte le chiavi con la crittografia Advanced Encryption Standard (AES) e le chiavi di crittografia a 256 bit. Quindi, archivia le chiavi crittografate con i dati crittografati, in modo che siano disponibili per decrittografare i dati del cluster on demand.

Quando si modifica la chiave KMS per il cluster, Aurora DSQL crittografa nuovamente la chiave del cluster esistente con la nuova chiave KMS.

Caching della chiave del cluster

Per evitare di chiamare AWS KMS per ogni operazione Aurora DSQL, Aurora DSQL memorizza nella cache le chiavi del cluster in testo semplice per ogni chiamante in memoria. Se Aurora DSQL riceve una richiesta per la chiave del cluster memorizzata nella cache dopo 15 minuti di inattività, invia una nuova richiesta per AWS KMS decrittografare la chiave del cluster. Questa chiamata acquisirà tutte le modifiche apportate alle politiche di accesso di AWS KMS key in AWS KMS o AWS Identity and Access Management (IAM) dopo l'ultima richiesta di decrittografia della chiave del cluster.

Autorizzazione all'uso del tuo AWS KMS key per Aurora DSQL

Se si utilizza una chiave gestita dal cliente nel proprio account per proteggere il cluster Aurora DSQL, è necessario che le policy su tale chiave forniscano ad Aurora DSQL l'autorizzazione per usarla per conto dell'utente.

L'utente ha il pieno controllo sulle policy di una chiave gestita dal cliente. Aurora DSQL non necessita di autorizzazioni aggiuntive per utilizzare l'impostazione predefinita per proteggere i Chiave di proprietà di AWS cluster Aurora DSQL nel tuo. Account AWS

Policy della chiave per una chiave gestita dal cliente

Quando si seleziona una chiave gestita dal cliente per proteggere un cluster Aurora DSQL, Aurora DSQL necessita dell'autorizzazione per utilizzarla per AWS KMS key conto del principale che effettua la selezione. Tale principale, un utente o un ruolo, deve disporre delle autorizzazioni richieste da Aurora DSQL. AWS KMS key È possibile fornire queste autorizzazioni in una policy della chiave o in una policy IAM.

Le autorizzazioni minime richieste da Aurora DSQL per una chiave gestita dal cliente sono:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(per e) kms: ReEncryptFrom kms: ReEncryptTo
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Ad esempio, la policy della chiave di esempio riportata di seguito fornisce solo le autorizzazioni necessarie. La policy ha i seguenti effetti:

- Consente ad Aurora DSQL di utilizzare Aurora DSQL AWS KMS key nelle operazioni crittografiche, ma solo quando agisce per conto dei responsabili dell'account che dispongono del permesso di utilizzare Aurora DSQL. Se le entità principali specificate nell'istruzione della policy non dispongono dell'autorizzazione per l'utilizzo di Aurora DSQL, la chiamata non riesce, anche quando proviene dal servizio Aurora DSQL.
- La chiave di condizione `kms:ViaService` consente le autorizzazioni solo quando la richiesta proviene da Aurora DSQL per conto delle entità principali elencate nell'istruzione della policy. Tali entità principali non possono chiamare direttamente queste operazioni.

- Fornisce agli AWS KMS key amministratori (utenti che possono assumere il ruolo) l'accesso in sola lettura a `db-team` AWS KMS key

Prima di utilizzare una politica chiave di esempio, sostituisci i principi di esempio con i principali effettivi del tuo. Account AWS

```
{
  "Sid": "Enable dsql IAM User Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:Encrypt",
    "kms:ReEncryptFrom",
    "kms:ReEncryptTo"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:dsql:ClusterId": "w4abucpbwuxx",
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
},
{
  "Sid": "Enable dsql IAM User Describe Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
}
```

Contesto di crittografia di Aurora DSQL

Un contesto di crittografia è un set di coppie chiave-valore che contiene dati arbitrari non segreti. Quando includi un contesto di crittografia in una richiesta di crittografia dei dati, associa AWS KMS crittograficamente il contesto di crittografia ai dati crittografati. lo stesso contesto di crittografia sia necessario per decrittografare i dati.

Aurora DSQL utilizza lo stesso contesto di crittografia in tutte le AWS KMS operazioni crittografiche. Se si utilizza una chiave gestita dal cliente per proteggere il cluster Aurora DSQL, è possibile utilizzare il contesto di crittografia per identificare l'utilizzo di tale chiave nei record e AWS KMS key nei log di controllo. Viene inoltre visualizzato come testo in chiaro nei log, ad esempio quelli di AWS CloudTrail.

Il contesto di crittografia può anche essere usato come una condizione per le autorizzazioni nelle policy.

Nelle sue richieste aAWS KMS, Aurora DSQL utilizza un contesto di crittografia con una coppia chiave-valore:

```
"encryptionContext": {
  "aws:dsql:ClusterId": "w4abucpbwuxx"
},
```

La coppia chiave-valore identifica il cluster che Aurora DSQL sta crittografando. La chiave è `aws:dsql:ClusterId`. Il valore è l'identificativo del cluster.

Monitoraggio dell'interazione di Aurora DSQL con AWS KMS

Se utilizzi una chiave gestita dal cliente per proteggere i tuoi cluster Aurora DSQL, puoi utilizzare i AWS CloudTrail log per tenere traccia delle richieste a cui Aurora DSQL invia per tuo conto. AWS KMS

Espandi le seguenti sezioni per scoprire come Aurora DSQL utilizza le AWS KMS operazioni e. `GenerateDataKey` `Decrypt`

`GenerateDataKey`

Quando si abilita la crittografia dei dati a riposo su un cluster, Aurora DSQL crea una chiave del cluster univoca. Invia una `GenerateDataKey` richiesta a AWS KMS che specifica il nome AWS KMS key per il cluster.

L'evento che registra l'operazione `GenerateDataKey` è simile a quello del seguente evento di esempio. L'utente è l'account di servizio di Aurora DSQL. I parametri includono l'Amazon Resource Name (ARN) diAWS KMS key, un identificatore di chiave che richiede una chiave a 256 bit e il contesto di crittografia che identifica il cluster.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dsql.amazonaws.com"
    },
    "eventTime": "2025-05-16T18:41:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "dsql.amazonaws.com",
    "userAgent": "dsql.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dsql:ClusterId": "w4abucpbwuxx"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
    },
    "responseElements": null,
    "requestID": "2da2dc32-d3f4-4d6c-8a41-aff27cd9a733",
    "eventID": "426df0a6-ba56-3244-9337-438411f826f4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "f88e0dd8-6057-4ce0-b77d-800448426d4e",
    "vpcEndpointId": "AWS Internal",
    "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
    "eventCategory": "Management"
}
```

Decrittografia

Quando si accede a un cluster Aurora DSQL crittografato, Aurora DSQL deve decrittografare la chiave del cluster per poter decrittografare le chiavi sottostanti nella gerarchia. Quindi decrittografa i dati nel cluster. Per decrittografare la chiave del cluster, Aurora DSQL invia una `Decrypt` richiesta a AWS KMS che specifica il nome del cluster. AWS KMS key

L'evento che registra l'operazione `Decrypt` è simile a quello del seguente evento di esempio. L'utente è il principale dell'utente Account AWS che accede al cluster. I parametri includono la chiave del cluster crittografata (come blob di testo cifrato) e il contesto di crittografia che identifica il cluster. AWS KMSricava l'ID di dal testo cifrato. AWS KMS key

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "dsql.amazonaws.com"
  },
  "eventTime": "2018-02-14T16:42:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "dsql.amazonaws.com",
  "userAgent": "dsql.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "encryptionContext": {
      "aws:dsql:ClusterId": "w4abucpbwuxx"
    },
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
  },
```

```
  "responseElements": null,
  "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
  "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "AWS Internal",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "sharedEventID": "d99f2dc5-b576-45b6-aa1d-3a3822edbeeb",
  "vpcEndpointId": "AWS Internal",
  "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
  "eventCategory": "Management"
}
```

Creazione di un cluster Aurora DSQL crittografato

Tutti i cluster Aurora DSQL sono crittografati a riposo. Per impostazione predefinita, i cluster utilizzano una chiave Chiave di proprietà di AWS gratuita oppure è possibile specificare una chiave personalizzata. AWS KMS Segui questi passaggi per creare il tuo cluster crittografato da Console di gestione AWS o da. AWS CLI

Console

Per creare un cluster crittografato in Console di gestione AWS

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Nel pannello di navigazione sul lato sinistro della console, scegli Cluster.
3. Scegli Crea cluster in alto a destra e seleziona Regione singola.
4. Nelle Impostazioni di crittografia del cluster, seleziona una delle seguenti opzioni.
   - Accetta le impostazioni predefinite per crittografare con un file senza Chiave di proprietà di AWS costi aggiuntivi.

- Seleziona Personalizza le impostazioni di crittografia (avanzate) per specificare una chiave KMS personalizzata. Quindi, cerca o inserisci l'ID o l'alias della tua chiave KMS. In alternativa, scegli Crea una AWS KMS chiave per creare una nuova chiave nella AWS KMS console.

5. Scegli Crea cluster.

Per confermare il tipo di crittografia per il cluster, vai alla pagina Cluster e seleziona l'ID del cluster per visualizzarne i dettagli. Esamina la scheda Impostazioni cluster: l'impostazione della chiave Cluster KMS mostra la chiave predefinita Aurora DSQL per i cluster che AWS utilizzano chiavi di proprietà o l'ID della chiave per altri tipi di crittografia.

> **Note**
>
> Se scegli di possedere e gestire la tua chiave, assicurati che la policy della chiave KMS sia impostata in modo appropriato. Per esempi e maggiori informazioni, consulta the section called "Policy della chiave per una chiave gestita dal cliente".

CLI

Per creare un cluster crittografato con l'impostazione predefinita Chiave di proprietà di AWS

- Utilizza il comando seguente per creare un cluster Aurora DSQL.

```
aws dsql create-cluster
```

Come illustrato nei seguenti dettagli di crittografia, lo stato di crittografia per il cluster è abilitato per impostazione predefinita e il tipo di crittografia predefinito è la chiave di proprietà di AWS. Il cluster è ora crittografato con la chiave predefinita di proprietà di AWS nell'account del servizio Aurora DSQL.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Come creare un cluster crittografato con la chiave gestita dal cliente

- Utilizza il comando seguente per creare un cluster Aurora DSQL, sostituendo l'ID della chiave scritto in rosso con l'ID della chiave gestita da te in qualità di cliente.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Come illustrato nei seguenti dettagli di crittografia, lo stato di crittografia per il cluster è abilitato per impostazione predefinita e il tipo di crittografia è la chiave KMS gestita dal cliente. Il cluster è ora crittografato con la chiave dell'utente.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Rimozione o aggiornamento di una chiave per il cluster Aurora DSQL

Puoi usare Console di gestione AWS o the AWS CLI per aggiornare o rimuovere le chiavi di crittografia sui cluster esistenti in Amazon Aurora DSQL. Se rimuovi una chiave senza sostituirla, Aurora DSQL utilizza l'impostazione predefinita Chiave di proprietà di AWS. Segui questi passaggi per aggiornare le chiavi di crittografia di un cluster esistente dalla console Aurora DSQL o dalla AWS CLI.

Console

Per aggiornare o rimuovere una chiave di crittografia in Console di gestione AWS

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Nel pannello di navigazione sul lato sinistro della console, scegli Cluster.
3. Nella visualizzazione a elenco, trova e seleziona la riga del cluster che desideri aggiornare.
4. Seleziona il menu Operazioni e quindi scegli Modifica.

5. Nelle Impostazioni di crittografia del cluster, scegli una delle seguenti opzioni per modificare le impostazioni di crittografia.

   - Se desideri passare da una chiave personalizzata a unaChiave di proprietà di AWS, deseleziona l'opzione Personalizza le impostazioni di crittografia (avanzate). Le impostazioni predefinite applicheranno e crittograferanno il cluster Chiave di proprietà di AWS gratuitamente.
   - Se desideri passare da una chiave KMS personalizzata a un'altra o da una Chiave di proprietà di AWS a una chiave KMS, seleziona l'opzione Personalizza le impostazioni di crittografia (avanzate) se non è già selezionata. Quindi, cerca e seleziona l'ID o l'alias della chiave che desideri utilizzare. In alternativa, scegli Crea una AWS KMS chiave per creare una nuova chiave nella AWS KMS console.

6. Scegli Save (Salva).

CLI

Gli esempi seguenti mostrano come utilizzare AWS CLI per aggiornare un cluster crittografato.

Per aggiornare un cluster crittografato con quello predefinito Chiave di proprietà di AWS

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

Lo `EncryptionStatus` della descrizione del cluster è impostato su `ENABLED` ed `EncryptionType` è `AWS_OWNED_KMS_KEY`.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Questo cluster è ora crittografato utilizzando l'impostazione predefinita Chiave di proprietà di AWS nell'account del servizio Aurora DSQL.

Come aggiornare un cluster crittografato con una chiave gestita dal cliente per Aurora DSQL

Aggiorna il cluster crittografato, come nell'esempio seguente:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

Lo `EncryptionStatus` della descrizione del cluster passa a `UPDATING` ed `EncryptionType` è `CUSTOMER_MANAGED_KMS_KEY`. Dopo che Aurora DSQL avrà terminato la propagazione della nuova chiave attraverso la piattaforma, lo stato di crittografia diventa `ENABLED`

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Se scegli di possedere e gestire la tua chiave, assicurati che la policy della chiave KMS sia impostata in modo appropriato. Per esempi e maggiori informazioni, consulta the section called "Policy della chiave per una chiave gestita dal cliente".

Considerazioni sulla crittografia con Aurora DSQL

- Aurora DSQL crittografa tutti i dati a riposo del cluster. Non è possibile disabilitare questa crittografia o crittografare solo alcuni elementi in un cluster.
- AWS Backupcrittografa i backup e tutti i cluster ripristinati da questi backup. È possibile crittografare i dati di backup AWS Backup utilizzando la chiave AWS proprietaria o una chiave gestita dal cliente.
- Per Aurora DSQL sono stati abilitati i seguenti stati di protezione:
  - Dati a riposo - Aurora DSQL crittografa tutti i dati statici su supporti di archiviazione persistenti

  - Dati in transito - Aurora DSQL crittografa tutte le comunicazioni tramite Transport Layer Security (TLS) per impostazione predefinita
- Quando passi a una chiave diversa, ti consigliamo di mantenere abilitata la chiave originale fino al completamento della transizione. AWSnecessita della chiave originale per decrittografare i dati prima di crittografare i dati con la nuova chiave. Il processo è completo quando il `encryptionStatus` del cluster è `ENABLED` e viene visualizzata la `kmsKeyArn` della nuova chiave gestita dal cliente.
- Quando si disabilita la chiave gestita dal cliente o si revoca l'accesso ad Aurora DSQL per l'utilizzo della chiave, lo stato del cluster diventa `IDLE`.
- L'API SQL di Amazon Aurora Console di gestione AWS e Amazon Aurora utilizzano termini diversi per i tipi di crittografia:
  - AWSConsole: nella console, vedrai `KMS` quando usi una chiave gestita dal cliente e `DEFAULT` quando usi un. Chiave di proprietà di AWS
  - API - L'API di Amazon Aurora DSQL utilizza `CUSTOMER_MANAGED_KMS_KEY` per le chiavi gestite dai clienti e `AWS_OWNED_KMS_KEY` per Chiavi di proprietà di AWS.
- Se non si specifica una chiave di crittografia durante la creazione del cluster, Aurora DSQL crittografa automaticamente i dati utilizzando il. Chiave di proprietà di AWS
- Puoi passare da una Chiave di proprietà di AWS chiave gestita dal cliente a una chiave gestita dal cliente in qualsiasi momento. Apporta questa modifica utilizzando Console di gestione AWSAWS CLI, o l'API SQL di Amazon Aurora.

Gestione delle identità e degli accessi per Aurora DSQL

AWS Identity and Access Management(IAM) è uno strumento Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle risorse. AWS Gli amministratori IAM controllano chi può essere autenticato (chi ha effettuato l'accesso) e autorizzato (chi dispone delle autorizzazioni) a utilizzare le risorse di Aurora DSQL. IAM è un software Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Argomenti

- Esempi di policy basate sull'identità per Amazon Aurora DSQL
- Risoluzione dei problemi di identità e accesso in Amazon Aurora DSQL

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (consulta Risoluzione dei problemi di identità e accesso in Amazon Aurora DSQL)
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (consulta Funzionamento di Amazon Aurora DSQL con IAM)
- Amministratore IAM: scrivi policy per gestire l'accesso (consulta Esempi di policy basate sull'identità per Amazon Aurora DSQL)

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/ Facebook Per ulteriori informazioni sull'accesso, consulta Come accedere all'Account AWS nella Guida per l'utente di Accedi ad AWS.

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente di IAM.

Account AWSutente root

Quando si crea unAccount AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali dell'utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Directory Service che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, si consiglia di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta Che cos'è il Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center.

Utenti e gruppi IAM

Un utente IAM è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta Richiedere agli utenti umani di utilizzare la federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo passando da un ruolo utente a un ruolo IAM (console) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta Non riesco ad assumere un ruolonella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso federato degli utenti, le autorizzazioni utente IAM temporanee, l'accesso tra account, l'accesso tra servizi e le applicazioni in esecuzione su Amazon. EC2 Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e collegandole a identità o risorse. AWS Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWSvaluta queste

politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy definiscono le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (incorporate direttamente in una singola identità) o policy gestite (policy standalone collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scegliere tra policy gestite e policy in linea nella Guida per l'utente di IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio specificare un'entità principale.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Altri tipi di policy

AWSsupporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in. AWS Organizations Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations.
- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta Politiche di controllo delle risorse (RCPs) nella Guida per l'AWS Organizationsutente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Funzionamento di Amazon Aurora DSQL con IAM

Prima di utilizzare IAM per gestire l'accesso ad Aurora DSQL è opportuno scoprire quali funzionalità di IAM sono disponibili per l'uso con questo servizio.

Funzionalità IAM utilizzabili con Amazon Aurora DSQL

Funzionalità IAM	Aurora DSQL supporta
Policy basate sull'identità	Sì
Policy basate su risorse	Sì
Operazioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione delle policy	Sì

Funzionalità IAM	Aurora DSQL supporta
ACLs	No
ABAC (tag nelle policy)	Sì
Credenziali temporanee	Sì
Autorizzazioni del principale	Sì
Ruoli di servizio	Sì
Ruoli collegati al servizio	Sì

Per avere una visione di alto livello di come Aurora DSQL e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, AWSconsulta i servizi che funzionano con IAM nella IAM User Guide.

Policy basate sull'identità per Aurora DSQL

Supporta le policy basate sull'identità: sì

Le policy basate sull'identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente di IAM.

Con le policy basate sull'identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta Guida di riferimento agli elementi delle policy JSON IAM nella Guida per l'utente IAM.

Esempi di policy basate sull'identità per Aurora DSQL

Per visualizzare esempi di policy basate sull'identità di Aurora DSQL, consulta Esempi di policy basate sull'identità per Amazon Aurora DSQL.

Policy basate sulle risorse all'interno di Aurora DSQL

Supporta le policy basate sulle risorse: sì

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy di bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. In una policy basata sulle risorse è obbligatorio specificare un'entità principale. I principali possono includere account, utenti, ruoli, utenti federati o servizi AWS. Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare i criteri gestiti AWS da IAM in un criterio basato sulle risorse.

Per informazioni su come creare e gestire policy basate sulle risorse per i cluster Aurora DSQL, consulta Policy basate sulle risorse per Aurora DSQL.

Operazioni di policy per Aurora DSQL

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy JSON per specificare chi ha accesso a cosa. AWS In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso in una policy. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di operazioni di Amazon Aurora DSQL, consulta Operazioni definite da Amazon Aurora DSQL nella Guida di riferimento all'autorizzazione del servizio.

Le operazioni delle policy in Aurora DSQL utilizzano il seguente prefisso prima dell'operazione:

```
dsql
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
      "dsql:action1",
```

```
        "dsql:action2"
]
```

Per visualizzare esempi di policy basate sull'identità di Aurora DSQL, consulta Esempi di policy basate sull'identità per Amazon Aurora DSQL.

Risorse relative alle policy per Aurora DSQL

Supporta le risorse relative alle policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Per le azioni che non supportano le autorizzazioni a livello di risorsa, si utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di risorse Aurora DSQL e relativi ARNs, consulta Resources Defined by Amazon Aurora DSQL nel Service Authorization Reference. Per informazioni sulle operazioni con cui è possibile specificare l'ARN di ogni risorsa, consulta Operazioni definite da Amazon Aurora DSQL.

Per visualizzare esempi di policy basate sull'identità di Aurora DSQL, consulta Esempi di policy basate sull'identità per Amazon Aurora DSQL.

Chiavi di condizione delle policy per Aurora DSQL

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio

uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di contesto delle condizioni AWS globali nella Guida per l'utente IAM.

Per un elenco delle chiavi di condizione di Aurora DSQL, consulta Chiavi di condizione per Amazon Aurora DSQL in Guida di riferimento all'autorizzazione del servizio. Per sapere con quali azioni e risorse è possibile utilizzare una chiave di condizione, consulta Operazioni definite da Amazon Aurora DSQL.

Per visualizzare esempi di policy basate sull'identità di Aurora DSQL, consulta Esempi di policy basate sull'identità per Amazon Aurora DSQL.

ACLs in Aurora SQL

Supporti ACLs: no

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con Aurora DSQL

Supporta ABAC (tag nelle policy): sì

Il controllo degli accessi basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base ad attributi chiamati tag. È possibile allegare tag a entità e AWS risorse IAM, quindi progettare politiche ABAC per consentire operazioni quando il tag del principale corrisponde al tag sulla risorsa.

Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Sì. Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per maggiori informazioni su ABAC, consulta Definizione delle autorizzazioni con autorizzazione ABAC nella Guida per l'utente di IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di

ABAC, consulta Utilizzo del controllo degli accessi basato su attributi (ABAC) nella Guida per l'utente di IAM.

Utilizzo di credenziali temporanee con Aurora DSQL

Supporta le credenziali temporanee: sì

Le credenziali temporanee forniscono l'accesso a breve termine alle AWS risorse e vengono create automaticamente quando si utilizza la federazione o si cambia ruolo. AWSconsiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza temporanee in IAM e Servizi AWS compatibili con IAM nella Guida per l'utente IAM.

Autorizzazioni dell'entità principale tra servizi per Aurora DSQL

Supporta l'inoltro delle sessioni di accesso (FAS): sì

Le sessioni di accesso inoltrato (FAS) utilizzano le autorizzazioni del principale che chiama e, in combinazione con la richiestaServizio AWS, Servizio AWS per effettuare richieste ai servizi downstream. Per i dettagli delle policy relative alle richieste FAS, consultare Forward access sessions.

Ruoli di servizio per Aurora DSQL

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per maggiori informazioni, consultare la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere la funzionalità di Aurora DSQL. Modifica i ruoli di servizio solo quando Aurora DSQL fornisce le indicazioni per farlo.

Ruoli collegati ai servizi per Aurora DSQL

Supporta i ruoli collegati ai servizi: sì

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati al servizio, ma non modificarle.

Per maggiori informazioni su come creare e gestire i ruoli collegati al servizio per Aurora DSQL, consulta Utilizzo dei ruoli collegati al servizio in Aurora DSQL.

Esempi di policy basate sull'identità per Amazon Aurora DSQL

Per impostazione predefinita, gli utenti e i ruoli non dispongono dell'autorizzazione per creare o modificare risorse di Aurora DSQL. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM (console) nella Guida per l'utente di IAM.

Per informazioni dettagliate sulle azioni e sui tipi di risorse definiti da Aurora DSQL, incluso il formato di ARNs per ogni tipo di risorsa, consulta Actions, Resources and Condition Keys for Amazon Aurora DSQL nel Service Authorization Reference.

Argomenti

- Best practice per le policy
- Utilizzo della console di Aurora DSQL
- Consentire agli utenti di visualizzare le loro autorizzazioni

Best practice per le policy

Le policy basate sull'identità determinano se qualcuno può creare, accedere o eliminare risorse di Aurora DSQL nell'account. Queste operazioni possono comportare costi aggiuntivi per l'Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Inizia con le politiche AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni. AWS Sono disponibili nel tuo. Account AWS Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti

specifiche per i tuoi casi d'uso. Per maggiori informazioni, consulta Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente di IAM.

- Applicazione delle autorizzazioni con privilegio minimo - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente di IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso - Per limitare l'accesso ad azioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specificoServizio AWS, ad esempioCloudFormation. Per maggiori informazioni, consultare la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente di IAM.
- Utilizzo dello strumento di analisi degli accessi IAM per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali - Lo strumento di analisi degli accessi IAM convalida le policy nuove ed esistenti in modo che aderiscano al linguaggio (JSON) della policy IAM e alle best practice di IAM. Lo strumento di analisi degli accessi IAM offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consultare Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente di IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consultare Protezione dell'accesso API con MFA nella Guida per l'utente di IAM.

Per maggiori informazioni sulle best practice in IAM, consulta Best practice di sicurezza in IAM nella Guida per l'utente di IAM.

Utilizzo della console di Aurora DSQL

Per accedere alla console di Amazon Aurora DSQL, è necessario disporre di un set di autorizzazioni minimo. Queste autorizzazioni devono consentire di elencare e visualizzare i dettagli sulle risorse Aurora DSQL presenti nel tuo. Account AWS Se crei una policy basata sull'identità più restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime per la console per gli utenti che effettuano chiamate solo verso o l'AWS CLIAPI. AWS Al contrario, è opportuno concedere l'accesso solo alle operazioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per garantire che utenti e ruoli possano ancora utilizzare la console Aurora DSQL, collega anche Aurora DSQL `AmazonAuroraDSQLConsoleFullAccess` o `AmazonAuroraDSQLReadOnlyAccess` AWS la policy gestita alle entità. Per maggiori informazioni, consulta Aggiunta di autorizzazioni a un utente nella Guida per l'utente di IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando l'API o a livello di codice. AWS CLI 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": "*"
        }
    ]
}
```

Risoluzione dei problemi di identità e accesso in Amazon Aurora DSQL

Utilizzare le informazioni seguenti per diagnosticare e risolvere i problemi comuni che possono verificarsi durante l'utilizzo di Aurora DSQL e IAM.

Argomenti

- Non si possiede l'autorizzazione a eseguire un'operazione in Aurora DSQL
- Non sono autorizzato a eseguire iam: PassRole
- Desidero consentire a persone esterne a me di accedere Account AWS alle mie risorse Aurora DSQL

Non si possiede l'autorizzazione a eseguire un'operazione in Aurora DSQL

Se si riceve un errore che indica che non si dispone dell'autorizzazione per eseguire un'operazione, le policy devono essere aggiornate in modo che sia consentito eseguire tale operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` tenta di utilizzare la console per visualizzare i dettagli relativi alla risorsa *`my-dsql-cluster`* ma non dispone delle autorizzazioni *`GetCluster`*.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-dsql-cluster`* utilizzando l'azione *`GetCluster`*.

Per ulteriore assistenza con l'accesso, contattare l'amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se si riceve un errore che indica che non si dispone dell'autorizzazione per eseguire l'operazione `iam:PassRole`, le policy devono essere aggiornate in modo che sia consentito trasmettere un ruolo ad Aurora DSQL.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in Aurora DSQL. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Desidero consentire a persone esterne a me di accedere Account AWS alle mie risorse Aurora DSQL

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per maggiori informazioni, consulta gli argomenti seguenti:

- Per capire se Aurora DSQL supporta queste funzionalità, consulta Funzionamento di Amazon Aurora DSQL con IAM.
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.

- Per scoprire come fornire l'accesso alle tue risorse a terze partiAccount AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (federazione delle identità) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente di IAM.

Policy basate sulle risorse per Aurora DSQL

Utilizza le policy basate sulle risorse per Aurora DSQL per limitare o concedere l'accesso ai cluster tramite documenti di policy JSON che si allegano direttamente alle risorse del cluster. Queste policy forniscono un controllo dettagliato su chi può accedere al cluster e in quali condizioni.

I cluster Aurora DSQL sono accessibili dalla rete Internet pubblica per impostazione predefinita, con l'autenticazione IAM come controllo di sicurezza principale. Le politiche basate sulle risorse consentono di aggiungere restrizioni di accesso, in particolare per bloccare l'accesso dalla rete Internet pubblica.

Le politiche basate sulle risorse funzionano insieme alle politiche basate sull'identità IAM. AWSvaluta entrambi i tipi di policy per determinare le autorizzazioni finali per qualsiasi richiesta di accesso al cluster. Per impostazione predefinita, i cluster Aurora DSQL sono accessibili all'interno di un account. Se un utente o un ruolo IAM dispone delle autorizzazioni Aurora DSQL, può accedere ai cluster senza alcuna policy basata sulle risorse allegata.

> **Note**
>
> Le modifiche alle policy basate sulle risorse alla fine sono coerenti e in genere entrano in vigore entro un minuto.

Per ulteriori informazioni sulle differenze tra le politiche basate sull'identità e le politiche basate sulle risorse, consulta Politiche basate sull'identità e politiche basate sulle risorse nella Guida per l'utente IAM.

Quando utilizzare le politiche basate sulle risorse

Le politiche basate sulle risorse sono particolarmente utili in questi scenari:

- Controllo degli accessi basato sulla rete: limita l'accesso in base al VPC o all'indirizzo IP da cui provengono le richieste o blocca completamente l'accesso pubblico a Internet. Usa chiavi condizionali come `aws:SourceVpc` e `aws:SourceIp` per controllare l'accesso alla rete.
- Più team o applicazioni: concedi l'accesso allo stesso cluster a più team o applicazioni. Anziché gestire le singole policy IAM per ogni principale, definisci le regole di accesso una sola volta nel cluster.
- Accesso condizionale complesso: controlla l'accesso in base a più fattori come gli attributi di rete, il contesto della richiesta e gli attributi utente. È possibile combinare più condizioni in un'unica politica.
- Governance della sicurezza centralizzata: consenti ai proprietari dei cluster di controllare l'accesso utilizzando una sintassi delle AWS policy familiare che si integra con le pratiche di sicurezza esistenti.

> **Note**
>
> L'accesso tra account non è ancora supportato per le policy basate sulle risorse di Aurora DSQL, ma sarà disponibile nelle versioni future.

Quando qualcuno tenta di connettersi al cluster Aurora DSQL, AWS valuta la policy basata sulle risorse come parte del contesto di autorizzazione, insieme a qualsiasi policy IAM pertinente, per determinare se la richiesta deve essere consentita o rifiutata.

Le politiche basate sulle risorse possono concedere l'accesso ai principali all'interno dello stesso account del cluster. AWS Per i cluster multiregionali, ogni cluster regionale dispone di una propria politica basata sulle risorse, che consente controlli di accesso specifici per regione quando necessario.

> **Note**
>
> Le chiavi del contesto delle condizioni possono variare tra le regioni (come VPC IDs).

Argomenti

- Creazione di cluster con politiche basate sulle risorse
- Aggiungere e modificare politiche basate sulle risorse per i cluster

- Visualizzazione delle politiche basate sulle risorse
- Rimozione delle politiche basate sulle risorse
- Esempi di politiche comuni basate sulle risorse
- Blocco dell'accesso pubblico con politiche basate sulle risorse in Aurora DSQL
- Operazioni dell'API Aurora DSQL e politiche basate sulle risorse

Creazione di cluster con politiche basate sulle risorse

È possibile allegare politiche basate sulle risorse durante la creazione di un nuovo cluster per garantire che i controlli di accesso siano attivi sin dall'inizio. Ogni cluster può avere una singola policy in linea collegata direttamente al cluster.

AWSConsole di gestione

Per aggiungere una policy basata sulle risorse durante la creazione del cluster

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Scegli Crea cluster.
3. Configura il nome del cluster, i tag e le impostazioni multiregionali in base alle esigenze.
4. Nella sezione Impostazioni del cluster, individua l'opzione di policy basata sulle risorse.
5. Attiva Aggiungi politica basata sulle risorse.
6. Inserisci il documento della policy nell'editor JSON. Ad esempio, per bloccare l'accesso pubblico a Internet:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
```

```
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Puoi utilizzare Edit statement o Add new statement per creare la tua politica.
8. Completa la configurazione rimanente del cluster e scegli Crea cluster.

AWSCLI

Usa il `--policy` parametro quando crei un cluster per allegare una policy in linea:

```
aws dsql create-cluster --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}'
```

AWSSDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
```

```
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}

response = client.create_cluster(
    policy=json.dumps(policy)
)

print(f"Cluster created: {response['identifier']}")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
    }
  }]
}
""";

CreateClusterRequest request = CreateClusterRequest.builder()
    .policy(policy)
    .build();

CreateClusterResponse response = client.createCluster(request);
```

```
System.out.println("Cluster created: " + response.identifier());
```

Aggiungere e modificare politiche basate sulle risorse per i cluster

AWSConsole di gestione

Per aggiungere una policy basata sulle risorse a un cluster esistente

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Scegli il tuo cluster dall'elenco dei cluster per aprire la pagina dei dettagli del cluster.
3. Scegli la scheda Autorizzazioni.
4. Nella sezione Politica basata sulle risorse, scegli Aggiungi politica.
5. Inserisci il tuo documento di policy nell'editor JSON. Puoi utilizzare Modifica dichiarazione o Aggiungi nuova dichiarazione per creare la tua politica.
6. Scegli Aggiungi policy.

Per modificare una politica esistente basata sulle risorse

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Scegli il tuo cluster dall'elenco dei cluster per aprire la pagina dei dettagli del cluster.
3. Scegli la scheda Autorizzazioni.
4. Nella sezione Politica basata sulle risorse, scegli Modifica.
5. Modifica il documento di policy nell'editor JSON. Puoi utilizzare Modifica dichiarazione o Aggiungi nuova dichiarazione per aggiornare la tua politica.
6. Scegli Save changes (Salva modifiche).

AWSCLI

Usa il `put-cluster-policy` comando per allegare una nuova politica o aggiornare una politica esistente su un cluster:

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
```

```
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

AWSSDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": {"aws:SourceVpc": "true"}
        }
    }]
}

response = client.put_cluster_policy(
    identifier='your_cluster_id',
    policy=json.dumps(policy)
)
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.PutClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "Null": {"aws:SourceVpc": "true"}
    }
  }]
}
""";

PutClusterPolicyRequest request = PutClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .policy(policy)
    .build();

client.putClusterPolicy(request);
```

Visualizzazione delle politiche basate sulle risorse

È possibile visualizzare le politiche basate sulle risorse allegate ai cluster per comprendere gli attuali controlli di accesso in vigore.

AWSConsole di gestione

Per visualizzare le politiche basate sulle risorse

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Scegli il tuo cluster dall'elenco dei cluster per aprire la pagina dei dettagli del cluster.
3. Scegli la scheda Autorizzazioni.

4. Visualizza la politica allegata nella sezione Politica basata sulle risorse.

AWSCLI

Usa il `get-cluster-policy` comando per visualizzare la politica basata sulle risorse di un cluster:

```
aws dsql get-cluster-policy --identifier your_cluster_id
```

AWSSDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

response = client.get_cluster_policy(
    identifier='your_cluster_id'
)

# Parse and pretty-print the policy
policy = json.loads(response['policy'])
print(json.dumps(policy, indent=2))
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyRequest;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyResponse;

DsqlClient client = DsqlClient.create();

GetClusterPolicyRequest request = GetClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

GetClusterPolicyResponse response = client.getClusterPolicy(request);
System.out.println("Policy: " + response.policy());
```

Rimozione delle politiche basate sulle risorse

È possibile rimuovere le politiche basate sulle risorse dai cluster per modificare i controlli di accesso.

> **Important**
>
> Quando rimuovi tutte le policy basate sulle risorse da un cluster, l'accesso sarà controllato interamente da policy basate sull'identità IAM.

AWSConsole di gestione

Per rimuovere una politica basata sulle risorse

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Scegli il tuo cluster dall'elenco dei cluster per aprire la pagina dei dettagli del cluster.
3. Scegli la scheda Autorizzazioni.
4. Nella sezione Politica basata sulle risorse, scegli Elimina.
5. Nella finestra di dialogo di conferma, digita **`confirm`** per confermare l'eliminazione.
6. Scegli Elimina.

AWSCLI

Usa il `delete-cluster-policy` comando per rimuovere una policy da un cluster:

```
aws dsql delete-cluster-policy --identifier your_cluster_id
```

AWSSDKs

Python

```
import boto3

client = boto3.client('dsql')
```

```
response = client.delete_cluster_policy(
    identifier='your_cluster_id'
)

print("Policy deleted successfully")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

DeleteClusterPolicyRequest request = DeleteClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

client.deleteClusterPolicy(request);
System.out.println("Policy deleted successfully");
```

Esempi di politiche comuni basate sulle risorse

Questi esempi mostrano modelli comuni per il controllo dell'accesso ai cluster Aurora DSQL. È possibile combinare e modificare questi modelli per soddisfare i requisiti di accesso specifici.

Blocca l'accesso pubblico a Internet

Questa policy blocca le connessioni ai cluster Aurora DSQL dalla rete Internet pubblica (non VPC). La policy non specifica da quale VPC i clienti possono connettersi, ma solo che devono connettersi da un VPC. Per limitare l'accesso a un VPC specifico, utilizzalo `aws:SourceVpc` con l'operatore delle `StringEquals` condizioni.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
```

```
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> Questo esempio serve solo `aws:SourceVpc` a verificare la presenza di connessioni VPC. Le chiavi `aws:VpcSourceIp` e `aws:SourceVpce` condition forniscono una granularità aggiuntiva ma non sono necessarie per il controllo di accesso di base basato solo su VPC.

Per fornire un'eccezione per ruoli specifici, utilizza invece questa politica:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromOutsideVPC",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        },
        "StringNotEquals": {
          "aws:PrincipalArn": [
```

```
            "arn:aws:iam::123456789012:role/ExceptionRole",
            "arn:aws:iam::123456789012:role/AnotherExceptionRole"
          ]
        }
      }
    }
  ]
}
```

Limita l'accesso all'AWSorganizzazione

Questa politica limita l'accesso ai responsabili all'interno di un'AWSorganizzazione:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster:mycluster",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "o-exampleorgid"
        }
      }
    }
  ]
}
```

Limita l'accesso a una specifica unità organizzativa

Questa politica limita l'accesso ai responsabili all'interno di una specifica unità organizzativa (OU) di un'AWSorganizzazione, fornendo un controllo più granulare rispetto all'accesso a livello di organizzazione:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster:mycluster",
      "Condition": {
        "StringNotLike": {
          "aws:PrincipalOrgPaths": "o-exampleorgid/r-examplerootid/ou-exampleouid/*"
        }
      }
    }
  ]
}
```

Politiche di cluster multiregionali

Per i cluster multiregionali, ogni cluster regionale mantiene la propria politica in materia di risorse, che consente controlli specifici per regione. Ecco un esempio con politiche diverse per regione:

politica us-east-1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
```

```
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

politica us-east-2:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> Le chiavi del contesto delle condizioni possono variare tra Regioni AWS (come VPC IDs).

Blocco dell'accesso pubblico con politiche basate sulle risorse in Aurora DSQL

Block Public Access (BPA) è una funzionalità che identifica e impedisce l'associazione di policy basate sulle risorse che garantiscono l'accesso pubblico ai cluster Aurora DSQL tra i tuoi account.

AWS Con BPA, puoi impedire l'accesso pubblico alle tue risorse Aurora DSQL. BPA esegue controlli durante la creazione o la modifica di una policy basata sulle risorse e aiuta a migliorare il livello di sicurezza con Aurora DSQL.

Il BPA utilizza il ragionamento automatico per analizzare l'accesso concesso dalla policy basata su risorse e avvisa l'utente se tali autorizzazioni vengono rilevate al momento della gestione di una policy basata su risorse. L'analisi verifica l'accesso a tutte le istruzioni della policy basata su risorse, alle azioni e al set di chiavi di condizione utilizzate nelle policy.

> **Important**
>
> BPA aiuta a proteggere le risorse impedendo che l'accesso pubblico venga concesso tramite le politiche basate sulle risorse direttamente collegate alle risorse Aurora DSQL, come i cluster. Oltre ad attivare il BPA, controlla attentamente le seguenti policy per verificare che non concedano l'accesso pubblico:
>
> - Politiche basate sull'identità collegate ai principali associati (ad esempio, ruoli IAM) AWS
> - Politiche basate sulle risorse collegate alle AWS risorse associate (ad esempio, AWS chiavi Key Management Service (KMS))

È necessario assicurarsi che il principale non includa una voce * o che una delle chiavi di condizione specificate limiti l'accesso dei principali alla risorsa. Se la policy basata sulle risorse concede l'accesso pubblico al cluster su più accountAWS, Aurora DSQL impedirà all'utente di creare o modificare la policy fino a quando le specifiche all'interno della policy non saranno corrette e considerate non pubbliche.

È possibile rendere una policy non pubblica specificando uno o più principi all'interno del blocco del `Principal`. Il seguente esempio di policy basata su risorse blocca l'accesso pubblico specificando due principali.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
```

```
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Inoltre, le policy che limitano l'accesso specificando determinate chiavi di condizione non sono considerate pubbliche. Oltre alla valutazione del principale specificato nella policy basata su risorse, vengono utilizzate le seguenti chiavi di condizione attendibili per completare la valutazione di una policy basata su risorse per l'accesso non pubblico:

- `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`

Inoltre, affinché una policy basata su risorse non sia pubblica, i valori del nome della risorsa Amazon (ARN) e le chiavi di stringa non devono contenere caratteri jolly o variabili. Se la propria policy basata su risorse utilizza la chiave `aws:PrincipalIsAWSService`, è necessario assicurarsi di aver impostato il valore della chiave su true.

La policy seguente limita l'accesso all'utente `Ben` nell'account specificato. La condizione rende il `Principal` vincolato e non lo considera pubblico.

```
{
```

```
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

L'esempio seguente di una policy basata su risorse non pubblica limita `sourceVPC` a utilizzare l'operatore `StringEquals`.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operazioni dell'API Aurora DSQL e politiche basate sulle risorse

Le policy basate sulle risorse in Aurora DSQL controllano l'accesso a specifiche operazioni API. Le seguenti sezioni elencano tutte le operazioni dell'API Aurora DSQL organizzate per categoria, con un'indicazione di quali supportano le politiche basate sulle risorse.

La colonna Supporta RBP indica se l'operazione dell'API è soggetta alla valutazione delle politiche basate sulle risorse quando una policy è collegata al cluster.

Etichetta APIs

Operazione API	Description	Supporta RBP
ListTagsForResource	Elenca i tag per una risorsa Aurora DSQL	Sì
TagResource	Aggiunge tag a una risorsa Aurora DSQL	Sì
UntagResource	Rimuove i tag da una risorsa Aurora DSQL	Sì

Gestione dei cluster APIs

Operazione API	Description	Supporta RBP
CreateCluster	Crea un nuovo cluster	No
DeleteCluster	Elimina un cluster	Sì
GetCluster	Recupera informazioni su un cluster	Sì
GetVpcEndpointServiceName	Recupera il nome del servizio endpoint VPC per un cluster	Sì
ListClusters	Elenca i cluster presenti nel tuo account	No
UpdateCluster	Aggiorna la configurazione di un cluster	Sì

Proprietà multiregionale APIs

Operazione API	Description	Supporta RBP
AddPeerCluster	Aggiunge un cluster peer a una configurazione multiregionale	Sì

Operazione API	Description	Supporta RBP
PutMultiRegionProperties	Imposta proprietà multiregionali per un cluster	Sì
PutWitnessRegion	Imposta la regione di riferimento per un cluster multiregionale	Sì

Politica basata sulle risorse APIs

Operazione API	Description	Supporta RBP
DeleteClusterPolicy	Elimina la politica basata sulle risorse da un cluster	Sì
GetClusterPolicy	Recupera la politica basata sulle risorse per un cluster	Sì
PutClusterPolicy	Crea o aggiorna la politica basata sulle risorse per un cluster	Sì

AWS Fault Injection ServiceAPIs

Operazione API	Description	Supporta RBP
InjectError	Inietta errori per i test di iniezione in caso di guasto	No

Backup e ripristino APIs

Operazione API	Description	Supporta RBP
GetBackupJob	Recupera informazioni su un job di backup	No

Operazione API	Description	Supporta RBP
GetRestoreJob	Recupera informazioni su un processo di ripristino	No
StartBackupJob	Avvia un processo di backup per un cluster	Sì
StartRestoreJob	Avvia un processo di ripristino da un backup	No
StopBackupJob	Interrompe un processo di backup in esecuzione	No
StopRestoreJob	Interrompe un processo di ripristino in esecuzione	No

Utilizzo dei ruoli collegati al servizio in Aurora DSQL

Aurora DSQL utilizza ruoli collegati ai servizi AWS Identity and Access Management (IAM). Un ruolo collegato al servizio è un tipo di ruolo IAM univoco collegato direttamente ad Aurora DSQL. I ruoli collegati ai servizi sono predefiniti da Aurora DSQL e includono tutte le autorizzazioni richieste dal servizio per chiamare Servizi AWS per conto del cluster Aurora DSQL.

I ruoli collegati al servizio semplificano la configurazione dei piani di dimensionamento perché permettono di evitare l'aggiunta manuale delle autorizzazioni necessarie. Quando si crea un cluster, Aurora DSQL crea automaticamente un ruolo collegato al servizio per conto dell'utente. È possibile eliminare il ruolo collegato al servizio solo dopo aver eliminato tutti i cluster. Così facendo, le risorse di Aurora DSQL restano protette, perché non è possibile rimuovere inavvertitamente le autorizzazioni necessarie per l'accesso alle risorse.

Per informazioni sugli altri servizi che supportano i ruoli collegati al servizio, consulta Servizi AWS che funzionano con IAM e cercare i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegliere Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato al servizio per tale servizio.

I ruoli collegati al servizio sono disponibili in tutte le Regioni Aurora DSQL supportate.

Autorizzazioni del ruolo collegato al servizio per Aurora DSQL

Aurora DSQL utilizza il ruolo collegato al servizio denominato: consente ad `AWSServiceRoleForAuroraDsql` Amazon Aurora DSQL di creare e gestire risorse per tuo conto. AWS Questo ruolo collegato ai servizi è collegato alle seguenti policy gestite: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato al servizio è necessario configurare le relative autorizzazioni. Potrebbe essere visualizzato il messaggio di errore seguente: `You don't have the permissions to create an Amazon Aurora DSQL service-linked role.` Se viene visualizzato questo messaggio, assicurarsi che le autorizzazioni seguenti siano abilitate:
>
> JSON
>
> ```
> {
>     "Version":"2012-10-17",
>     "Statement": [
>         {
>             "Action": ["dsql:CreateCluster"],
>             "Resource": [
>                 "arn:aws:dsql:us-east-1:*:cluster/*",
>                 "arn:aws:dsql:us-east-2:*:cluster/*"
>             ],
>             "Effect": "Allow"
>         }
>     ]
> }
> ```
>
> Per maggiori informazioni, consulta Autorizzazioni del ruolo collegato al servizio.

Creare un ruolo collegato al servizio

Non è necessario creare manualmente un ruolo collegato ai DSQLService LinkedRolePolicy servizi Aurora. Aurora DSQL crea il ruolo collegato al servizio per conto dell'utente. Se il ruolo DSQLService LinkedRolePolicy collegato al servizio Aurora è stato eliminato dall'account, Aurora DSQL crea il ruolo quando si crea un nuovo cluster Aurora DSQL.

Modifica di un ruolo collegato al servizio

Aurora DSQL non consente di modificare il ruolo collegato al servizio Aurora. DSQLService LinkedRolePolicy Dopo aver creato un ruolo collegato al servizio, non è possibile modificarne il nome, perché potrebbero farvi riferimento diverse entità. Tuttavia, puoi modificare la descrizione del ruolo utilizzando la console IAM, la AWS Command Line Interface (AWS CLI) o l'API IAM.

Eliminazione di un ruolo collegato al servizio

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, consigliamo di eliminare il ruolo. In questo modo non sarà più presente un'entità non utilizzata che non viene monitorata e gestita attivamente.

Prima di poter eliminare un ruolo legato a un servizio per un account, è necessario arrestare ed eliminare qualsiasi cluster nell'account.

Puoi utilizzare la console IAMAWS CLI, l'o l'API IAM per eliminare un ruolo collegato al servizio. Per maggiori informazioni, consulta Creazione di un ruolo collegato al servizio nella Guida per l'utente di IAM.

Regioni supportate per i ruoli collegati al servizio di Aurora DSQL

Aurora DSQL supporta l'utilizzo di ruoli collegati al servizio in tutte le Regioni in cui il servizio è disponibile. Per maggiori informazioni, consulta Regioni ed endpoint di AWS.

Utilizzo di chiavi di condizione IAM con Amazon Aurora DSQL

Quando si concedono le autorizzazioni in Aurora DSQL, è possibile specificare le condizioni che determinano il modo in cui una policy di autorizzazione viene applicata. Di seguito sono illustrati esempi di come è possibile utilizzare le chiavi di condizione nelle policy di autorizzazioni IAM di Aurora DSQL.

Esempio 1: concedere l'autorizzazione per creare un cluster in uno specifico Regione AWS

Le seguenti policy concedono l'autorizzazione a creare cluster nelle Regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti orientali (Ohio). Questa policy utilizza l'ARN della risorsa per limitare le Regioni consentite, quindi Aurora DSQL può creare cluster solo se tale ARN è specificato nella sezione `Resource` della policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Esempio 2: concedere l'autorizzazione a creare un cluster multiregionale in s specifici Regione AWS

Le seguenti policy concedono l'autorizzazione a creare cluster multi-Regione nelle Regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti orientali (Ohio). Questa policy utilizza l'ARN della risorsa per limitare le Regioni consentite, quindi Aurora DSQL può creare cluster multi-Regione solo se questo ARN è specificato nella sezione `Resource` della policy. Tenere presente che la creazione di cluster multi-Regione richiede anche le autorizzazioni `PutMultiRegionProperties`, `PutWitnessRegion` e `AddPeerCluster` in ciascuna Regione specificata.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "dsql:CreateCluster",
          "dsql:PutMultiRegionProperties",
          "dsql:PutWitnessRegion",
          "dsql:AddPeerCluster"
        ],
        "Resource": [
           "arn:aws:dsql:us-east-1:123456789012:cluster/*",
           "arn:aws:dsql:us-east-2:123456789012:cluster/*"
        ]
      }
    ]
}
```

Esempio 3: concessione dell'autorizzazione per creare un cluster multi-Regione con una Regione testimone specifica

La seguente policy utilizza una chiave di condizione `dsql:WitnessRegion` di Aurora DSQL e consente a un utente di creare cluster multi-Regione con una Regione testimone negli Stati Uniti occidentali (Oregon). Se non si specifica la condizione `dsql:WitnessRegion`, è possibile utilizzare qualsiasi Regione come Regione testimone.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": [
                        "us-west-2"
                    ]
                }
            }
        }
    ]
}
```

Risposta agli incidenti in Amazon Aurora DSQL

Per AWS, la sicurezza ha la massima priorità. Come parte del modello di responsabilità condivisa del AWS cloud, AWS gestisce un data center, una rete e un'architettura software che soddisfa i requisiti delle organizzazioni più sensibili alla sicurezza. AWSè responsabile di qualsiasi risposta agli incidenti relativi al servizio SQL di Amazon Aurora. Inoltre, come AWS cliente, condividi la responsabilità di mantenere la sicurezza nel cloud. Ciò significa che puoi controllare la sicurezza che scegli di implementare dagli AWS strumenti e dalle funzionalità a cui hai accesso. Inoltre, l'utente è responsabile della risposta agli incidenti relativi alla sua parte del modello di responsabilità condivisa.

Stabilendo una linea di base di sicurezza che soddisfi gli obiettivi delle applicazioni eseguite nel cloud, l'utente è in grado di rilevare deviazioni a cui è possibile rispondere. Per comprendere l'impatto che la risposta agli incidenti e le tue scelte hanno sugli obiettivi aziendali, ti invitiamo a consulta le seguenti risorse:

- AWSGuida alla risposta agli incidenti di sicurezza
- AWSLe migliori pratiche per la sicurezza, l'identità e la conformità
- White paper sulla prospettiva di sicurezza del AWS Cloud Adoption Framework (CAF)

Amazon GuardDuty è un servizio gestito di rilevamento delle minacce che monitora continuamente i comportamenti dannosi o non autorizzati per aiutare i clienti a proteggere i carichi di lavoro Account

AWS e identificare attività potenzialmente sospette prima che si trasformino in un incidente. Monitora attività come chiamate API insolite o implementazioni potenzialmente non autorizzate, indicando possibili compromissioni di account o risorse o ricognizioni da parte di malintenzionati. Ad esempio, Amazon GuardDuty è in grado di rilevare attività sospette in Amazon Aurora APIs DSQL, come un utente che accede da una nuova posizione e crea un nuovo cluster.

Convalida della conformità per Amazon Aurora DSQL

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWSconformità Programmi di .

È possibile scaricare report di audit di terze parti utilizzandoAWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzoServizi AWS, consulta la Documentazione AWS sulla sicurezza.

Resilienza in Amazon Aurora DSQL

L'infrastruttura AWS globale è costruita attorno a zone di disponibilità (Regioni AWSAZ). Regioni AWSforniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità è possibile progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali. Aurora DSQL è progettato in modo da poter sfruttare l'infrastruttura AWS regionale fornendo al contempo la massima disponibilità del database. Per impostazione predefinita, i cluster a Regione singola in Aurora DSQL offrono una disponibilità Multi-AZ, che offre tolleranza ai principali guasti dei componenti e alle interruzioni dell'infrastruttura che potrebbero influire sull'accesso a una zona di disponibilità completa. I cluster multi-Regione offrono tutti i vantaggi della resilienza Multi-AZ, pur garantendo una disponibilità del database estremamente costante, anche nei casi in cui la Regione AWS dovesse risultare inaccessibile ai client delle applicazioni.

Per ulteriori informazioni sulle Regioni AWS zone di disponibilità, consulta AWSGlobal Infrastructure.

Oltre all'infrastruttura AWS globale, Aurora DSQL offre diverse funzionalità per supportare le esigenze di resilienza e backup dei dati.

Backup e ripristino

Aurora DSQL supporta il backup e il ripristino con. Console di backup AWS È possibile eseguire un backup e un ripristino completi per i cluster a Regione singola e multi-Regione. Per maggiori informazioni, consultare Backup e ripristino per Amazon Aurora DSQL.

Replica

In base alla progettazione, Aurora DSQL esegue il commit di tutte le transazioni di scrittura in un registro delle transazioni distribuito e replica in modo sincrono tutti i dati di registro impegnati nelle repliche di archiviazione degli utenti in tre repliche. AZs I cluster multi-Regione offrono funzionalità complete di replica interregionale tra Regioni di lettura e scrittura.

Una Regione testimone designata supporta le scritture relative ai soli log delle transazioni e non consuma spazio di archiviazione. Le Regioni testimone non dispongono di un endpoint. Ciò significa che le Regioni testimone archiviano solo i registri delle transazioni crittografati, non richiedono alcuna amministrazione o configurazione e non sono accessibili dagli utenti.

I log delle transazioni e l'archiviazione degli utenti di Aurora DSQL sono distribuiti con tutti i dati presentati agli elaboratori delle query di Aurora DSQL come un unico volume logico. Aurora DSQL divide, unisce e replica automaticamente i dati in base all'intervallo di chiavi primarie del database e ai modelli di accesso. Aurora DSQL dimensiona automaticamente le repliche di lettura, sia verso l'alto che verso il basso, in base alla frequenza di accesso in lettura.

Le repliche dell'archiviazione del cluster sono distribuite su un parco di sistemi di archiviazione multi-tenant. Se un componente o un'AZ vengono danneggiati, Aurora DSQL reindirizza automaticamente l'accesso ai componenti sopravvissuti e ripara in modo asincrono le repliche mancanti. Una volta che Aurora DSQL corregge le repliche danneggiate, le aggiunge automaticamente al quorum di archiviazione e le rende disponibili per il cluster.

Elevata disponibilità

Per impostazione predefinita, i cluster a Regione singola e multi-Regione in Aurora DSQL sono attivi e non è necessario effettuare manualmente il provisioning, configurare o riconfigurare alcun cluster. Aurora DSQL automatizza completamente il ripristino del cluster, eliminando la necessità

delle tradizionali operazioni di failover primario-secondario. La replica è sempre sincrona e viene eseguita in più parti AZs, quindi non vi è alcun rischio di perdita dei dati dovuta al ritardo della replica o al failover su un database secondario asincrono durante il ripristino in caso di errore.

I cluster a regione singola forniscono un endpoint ridondante Multi-AZ che abilita automaticamente l'accesso simultaneo con una forte coerenza dei dati su tre. AZs Ciò significa che le repliche di storage degli utenti su ognuna di queste tre applicazioni restituiscono AZs sempre lo stesso risultato a uno o più lettori e sono sempre disponibili per ricevere scritture. Questa forte coerenza e resilienza Multi-AZ sono disponibili in tutte le Regioni per i cluster multi-Regione di Aurora DSQL. Ciò significa che i cluster multi-Regione forniscono due endpoint regionali fortemente coerenti, in modo che i client possano leggere o scrivere indiscriminatamente in entrambe le Regioni senza ritardi di replica al momento del commit.

Aurora DSQL offre una disponibilità del 99,99% per i cluster a Regione singola e del 99,999% per i cluster multi-Regione.

Test di iniezione di guasti

Amazon Aurora DSQL si integra con AWS Fault Injection Service (AWS FIS), un servizio completamente gestito per l'esecuzione di esperimenti di iniezione di errori controllati per migliorare la resilienza di un'applicazione. UtilizzandoAWS FIS, puoi:

- Creare modelli di esperimenti che definiscono scenari di errore specifici
- Eseguire l'iniezione di guasti (elevati tassi di errore di connessione al cluster) per convalidare i meccanismi di gestione e ripristino degli errori delle applicazioni
- Verifica il comportamento delle applicazioni in più regioni per convalidare lo spostamento del traffico delle applicazioni tra un momento e l'altro Regioni AWS quando Regione AWS si verificano alti tassi di errore di connessione

Ad esempio, in un cluster multi-Regione che copre gli Stati Uniti orientali (Virginia settentrionale) e gli Stati Uniti orientali (Ohio), è possibile eseguire un esperimento negli Stati Uniti orientali (Ohio) per verificare gli errori in tali aree mentre gli Stati Uniti orientali (Virginia settentrionale) continuano le normali operazioni. Questo test controllato consente di identificare e risolvere potenziali problemi prima che influiscano sui carichi di lavoro di produzione.

Per un elenco completo delle azioni AWS FIS supportate, consulta gli obiettivi delle azioni nella guida per l'AWS FISutente.

Per informazioni sulle azioni DSQL di Amazon Aurora disponibili in, AWS FIS consulta il riferimento alle azioni DSQL di Aurora nella Guida per l'utente. AWS FIS

Per iniziare a eseguire esperimenti di iniezione di guasti, consulta Pianificazione degli esperimenti di AWS FIS nella Guida per l'utente di AWS FIS.

Sicurezza dell'infrastruttura in Amazon Aurora DSQL

In quanto servizio gestito, Amazon Aurora DSQL è protetto dalle procedure di sicurezza di rete AWS globali descritte nelle Best Practices for Security, Identity and Compliance.

Si utilizzano chiamate API AWS pubblicate per accedere ad Aurora DSQL attraverso la rete. I client devono supportare Transport Layer Security (TLS) 1.2 o versioni successive. I client devono, inoltre, supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa è possibile utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Gestione e connessione ai cluster SQL di Amazon Aurora tramite AWS PrivateLink

Con AWS PrivateLink Amazon Aurora DSQL, puoi effettuare il provisioning degli endpoint Amazon VPC di interfaccia (endpoint di interfaccia) nel tuo Amazon Virtual Private Cloud. Questi endpoint sono accessibili direttamente dalle applicazioni locali tramite Amazon VPC Direct Connect e/o in un altro modo di peering Regione AWS tramite Amazon VPC. Utilizzando AWS PrivateLink e interfacciando gli endpoint, puoi semplificare la connettività di rete privata dalle tue applicazioni ad Aurora DSQL.

Le applicazioni all'interno di Amazon VPC possono accedere ad Aurora DSQL utilizzando gli endpoint di interfaccia Amazon VPC senza richiedere indirizzi IP pubblici.

Gli endpoint dell'interfaccia sono rappresentati da una o più interfacce di rete elastiche (ENIs) a cui vengono assegnati indirizzi IP privati dalle sottoreti del tuo Amazon VPC. Le richieste ad Aurora DSQL tramite endpoint di interfaccia rimangono sulla rete. AWS Per maggiori informazioni su come

connettere il VPC Amazon alla rete on-premises, consulta la Guida per l'utente di Direct Connect e la Guida per l'utente della VPN di AWS Site-to-Site VPN.

Per informazioni generali sugli endpoint di interfaccia, consulta Accedere a un AWS servizio utilizzando un endpoint Amazon VPC con interfaccia nella AWS PrivateLinkGuida per l'utente.

Tipi di endpoint Amazon VPC per Aurora DSQL

Aurora DSQL richiede due diversi tipi di endpoint. AWS PrivateLink

1. Endpoint di gestione - Questo endpoint viene utilizzato per operazioni amministrative, come `get`, `create`, `update`, `delete` e `list` sui cluster Aurora DSQL. Consulta Gestione dei cluster Aurora DSQL utilizzando AWS PrivateLink.
2. Endpoint di connessione - Questo endpoint viene utilizzato per la connessione ai cluster Aurora DSQL tramite client PostgreSQL. Per informazioni, consulta Connessione ai cluster Aurora DSQL tramite AWS PrivateLink.

Considerazioni sull'utilizzo AWS PrivateLink per Aurora DSQL

Le considerazioni relative ad Amazon VPC valgono per AWS PrivateLink Aurora DSQL. Per ulteriori informazioni, consulta Accedere a un AWS servizio utilizzando un endpoint e AWS PrivateLinkquote VPC di interfaccia nella Guida. AWS PrivateLink

Gestione dei cluster Aurora DSQL utilizzando AWS PrivateLink

È possibile utilizzare AWS Command Line Interface o AWS Software Development Kit (SDKs) per gestire i cluster Aurora DSQL tramite gli endpoint dell'interfaccia Aurora DSQL.

Creazione di un endpoint Amazon VPC

Per creare un endpoint di interfaccia Amazon VPC, consulta Creare un endpoint Amazon VPC nella Guida. AWS PrivateLink

```
aws ec2 create-vpc-endpoint \
--region region \
--service-name com.amazonaws.region.dsql \
--vpc-id your-vpc-id \
--subnet-ids your-subnet-id \
--vpc-endpoint-type Interface \
```

```
--security-group-ids client-sg-id \
```

Per utilizzare il nome DNS regionale predefinito per le richieste API Aurora DSQL, non disabilitare il DNS privato quando si crea l'endpoint dell'interfaccia Aurora DSQL. Quando il DNS privato è abilitato, le richieste al servizio Aurora DSQL effettuate direttamente dal tuo Amazon VPC verranno risolte automaticamente nell'indirizzo IP privato dell'endpoint Amazon VPC, anziché nel nome DNS pubblico. Quando il DNS privato è abilitato, le richieste di Aurora DSQL effettuate direttamente dal tuo Amazon VPC verranno risolte automaticamente nell'endpoint Amazon VPC.

Se il DNS privato non è abilitato, utilizzate i `--endpoint-url` parametri `--region` and con AWS CLI i comandi per gestire i cluster Aurora DSQL tramite gli endpoint dell'interfaccia Aurora DSQL.

Recupero dell'elenco dei cluster tramite un URL di endpoint

Nell'esempio seguente, sostituisci il nome DNS Regione AWS `us-east-1` e il nome DNS dell'`vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com`ID dell'endpoint Amazon VPC con le tue informazioni.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-
east-1.vpce.amazonaws.com list-clusters
```

Operazioni API

Fai riferimento alla Guida di riferimento delle API di Aurora DSQL per la documentazione sulla gestione delle risorse in Aurora DSQL.

Gestione delle policy degli endpoint

Testando e configurando accuratamente le policy degli endpoint Amazon VPC, puoi contribuire a garantire che il cluster Aurora DSQL sia sicuro, conforme e allineato ai requisiti di governance e controllo degli accessi specifici della tua organizzazione.

Esempio: policy di accesso ad Aurora DSQL completa

Le seguenti policy garantiscono l'accesso completo a tutte le azioni e le risorse di Aurora DSQL tramite l'endpoint Amazon VPC specificato.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxxx \
```

```
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

Esempio: policy di accesso ad Aurora DSQL con restrizioni

La seguente policy consente solo queste operazioni su Aurora DSQL.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Tutte le altre operazioni Aurora DSQL vengono negate.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Connessione ai cluster Aurora DSQL tramite AWS PrivateLink

Una volta che l'AWS PrivateLinkendpoint è configurato e attivo, puoi connetterti al tuo cluster Aurora DSQL utilizzando un client PostgreSQL. Le istruzioni di connessione riportate di seguito descrivono i passaggi per creare il nome host corretto per la connessione tramite l'endpoint. AWS PrivateLink

Impostazione di un endpoint di connessione AWS PrivateLink

Fase 1: recupero del nome del servizio del proprio cluster

Quando si crea un AWS PrivateLink endpoint per la connessione al cluster, è innanzitutto necessario recuperare il nome del servizio specifico del cluster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Risposta di esempio

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

Il nome del servizio include un identificatore, come `dsql-fnh4` nell'esempio. Questo identificatore è necessario anche per creare il nome host per la connessione al cluster.

AWS SDK for Python (Boto3)

```
import boto3

dsql_client = boto3.client('dsql', region_name='us-east-1')
response = dsql_client.get_vpc_endpoint_service_name(
    identifier='your-cluster-id'
)
service_name = response['serviceName']
print(f"Service Name: {service_name}")
```

AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameRequest;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameResponse;

String region = "us-east-1";
String clusterId = "your-cluster-id";

DsqlClient dsqlClient = DsqlClient.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

GetVpcEndpointServiceNameResponse response = dsqlClient.getVpcEndpointServiceName(
    GetVpcEndpointServiceNameRequest.builder()
        .identifier(clusterId)
        .build()
);
String serviceName = response.serviceName();
System.out.println("Service Name: " + serviceName);
```

Fase 2: creazione dell'endpoint Amazon VPC

Utilizzando il nome del servizio ottenuto nel passaggio precedente, crea un endpoint Amazon VPC.

> **Important**
>
> Le istruzioni di connessione riportate di seguito funzionano solo per la connessione ai cluster quando il DNS privato è abilitato. Non utilizzare il flag `--no-private-dns-enabled` durante la creazione dell'endpoint, poiché ciò impedirà il corretto funzionamento delle istruzioni di connessione riportate di seguito. Se di disabilita il DNS privato, sarà necessario creare il proprio record DNS privato con wildcard che punti all'endpoint creato.

AWS CLI

```
aws ec2 create-vpc-endpoint \
    --region us-east-1 \
    --service-name service-name-for-your-cluster \
    --vpc-id your-vpc-id \
```

```
    --subnet-ids subnet-id-1 subnet-id-2  \
    --vpc-endpoint-type Interface \
    --security-group-ids security-group-id
```

Risposta di esempio

```
{
    "VpcEndpoint": {
        "VpcEndpointId": "vpce-0123456789abcdef0",
        "VpcEndpointType": "Interface",
        "VpcId": "vpc-0123456789abcdef0",
        "ServiceName": "com.amazonaws.us-east-1.dsql-fnh4",
        "State": "pending",
        "RouteTableIds": [],
        "SubnetIds": [
            "subnet-0123456789abcdef0",
            "subnet-0123456789abcdef1"
        ],
        "Groups": [
            {
                "GroupId": "sg-0123456789abcdef0",
                "GroupName": "default"
            }
        ],
        "PrivateDnsEnabled": true,
        "RequesterManaged": false,
        "NetworkInterfaceIds": [
            "eni-0123456789abcdef0",
            "eni-0123456789abcdef1"
        ],
        "DnsEntries": [
            {
                "DnsName": "*.dsql-fnh4.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            }
        ],
        "CreationTimestamp": "2025-01-01T00:00:00.000Z"
    }
}
```

SDK for Python

```
import boto3
```

```
ec2_client = boto3.client('ec2', region_name='us-east-1')
response = ec2_client.create_vpc_endpoint(
    VpcEndpointType='Interface',
    VpcId='your-vpc-id',
    ServiceName='com.amazonaws.us-east-1.dsql-fnh4',  # Use the service name from
 previous step
    SubnetIds=[
        'subnet-id-1',
        'subnet-id-2'
    ],
    SecurityGroupIds=[
        'security-group-id'
    ]
)

vpc_endpoint_id = response['VpcEndpoint']['VpcEndpointId']
print(f"VPC Endpoint created with ID: {vpc_endpoint_id}")
```

SDK for Java 2.x

Usa un URL endpoint per Aurora DSQL APIs

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointRequest;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointResponse;
import software.amazon.awssdk.services.ec2.model.VpcEndpointType;

String region = "us-east-1";
String serviceName = "com.amazonaws.us-east-1.dsql-fnh4";  // Use the service name
 from previous step
String vpcId = "your-vpc-id";

Ec2Client ec2Client = Ec2Client.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

CreateVpcEndpointRequest request = CreateVpcEndpointRequest.builder()
    .vpcId(vpcId)
    .serviceName(serviceName)
    .vpcEndpointType(VpcEndpointType.INTERFACE)
```

```
    .subnetIds("subnet-id-1", "subnet-id-2")
    .securityGroupIds("security-group-id")
    .build();

CreateVpcEndpointResponse response = ec2Client.createVpcEndpoint(request);
String vpcEndpointId = response.vpcEndpoint().vpcEndpointId();
System.out.println("VPC Endpoint created with ID: " + vpcEndpointId);
```

Configurazione aggiuntiva durante la connessione Direct Connect tramite peering Amazon VPC

Potrebbero essere necessarie alcune configurazioni aggiuntive per connettersi ai cluster Aurora DSQL utilizzando un endpoint di AWS PrivateLink connessione da dispositivi locali tramite peering Amazon VPC o. Direct Connect Questa configurazione non è necessaria se l'applicazione è in esecuzione nello stesso Amazon VPC dell'endpoint. AWS PrivateLink Le voci DNS private create sopra non verranno risolte correttamente al di fuori dell'Amazon VPC dell'endpoint, ma puoi creare i tuoi record DNS privati che si risolvono nell'endpoint di connessione. AWS PrivateLink

Crea un record DNS CNAME privato che punti al nome di dominio completo dell'endpoint. AWS PrivateLink Il nome di dominio del record DNS creato deve essere costruito con i seguenti componenti:

1. L'identificatore del servizio dal nome del servizio. Ad esempio: `dsql-fnh4`
2. Il Regione AWS

Crea il record DNS CNAME con un nome di dominio nel seguente formato: `*.`*`service-identifier`*`.`*`region`*`.on.aws`

Il formato del nome di dominio è importante per due motivi:

1. Il nome host utilizzato per connettersi ad Aurora DSQL deve corrispondere al certificato del server di Aurora DSQL quando si utilizza la modalità SSL. `verify-full` Ciò garantisce il massimo livello di sicurezza della connessione.
2. Aurora DSQL utilizza la parte relativa all'ID del cluster del nome host utilizzata per connettersi ad Aurora DSQL per identificare il cluster di connessione.

Se non è possibile creare record DNS privati, è comunque possibile connettersi ad Aurora DSQL. Per informazioni, consulta Connessione a un cluster Aurora DSQL utilizzando un AWS PrivateLink endpoint senza DNS privato.

Connessione a un cluster Aurora DSQL utilizzando un endpoint di connessione AWS PrivateLink

Una volta che l'AWS PrivateLinkendpoint è configurato e attivo (verifica che lo `State` sia`available`), puoi connetterti al tuo cluster Aurora DSQL utilizzando un client PostgreSQL. Per istruzioni sull'uso di AWS SDKs, puoi seguire le guide in Programmazione con Aurora DSQL. È necessario modificare l'endpoint del cluster in modo che corrisponda al formato del nome host.

Costruzione del nome host

Il nome host per la connessione è AWS PrivateLink diverso dal nome host DNS pubblico. È necessario costruirlo utilizzando i seguenti componenti.

1. `Your-cluster-id`
2. L'identificatore del servizio dal nome del servizio. Ad esempio: `dsql-fnh4`
3. Il. Regione AWS Ad esempio: `us-east-1`

Utilizzare il seguente formato: *`cluster-id`*`.`*`service-identifier`*`.`*`region`*`.on.aws`

Esempio: connessione tramite PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export SERVICE_IDENTIFIER=dsql-fnh4  # This should match the identifier in your service
 name

# Construct the hostname
export HOSTNAME="$CLUSTERID.$SERVICE_IDENTIFIER.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $HOSTNAME)

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Connessione a un cluster Aurora DSQL utilizzando un AWS PrivateLink endpoint senza DNS privato

Le istruzioni di connessione riportate sopra si basano su record DNS privati. Se la tua applicazione è in esecuzione nello stesso Amazon VPC dell'AWS PrivateLinkendpoint, i record DNS vengono creati automaticamente. In alternativa, se ti connetti da dispositivi locali tramite peering Amazon VPC oppureDirect Connect, puoi creare i tuoi record DNS privati. Tuttavia, la configurazione dei record DNS non è sempre possibile a causa delle restrizioni di rete imposte dai team di sicurezza. Se l'applicazione deve connettersi tramite Direct Connect o da un Amazon VPC peered e la configurazione dei record DNS non è possibile, puoi comunque connetterti ad Aurora DSQL.

Aurora DSQL utilizza la parte relativa all'ID del cluster del nome host per identificare il cluster di connessione, ma se la configurazione del record DNS non è possibile, Aurora DSQL supporta la specificazione del cluster di destinazione utilizzando l'opzione di connessione. `amzn-cluster-id` Con questa opzione, è possibile utilizzare il nome di dominio completo dell'AWS PrivateLinkendpoint come nome host durante la connessione.

> **Important**
>
> Quando ci si connette con il nome di dominio o l'indirizzo IP completamente qualificato dell'AWS PrivateLinkendpoint, la modalità SSL non è supportata. `verify-full` Per questo motivo, è preferibile configurare un DNS privato.

Esempio: specificazione dell'opzione di connessione con l'ID del cluster utilizzando PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export HOSTNAME=vpce-04037adb76c111221-d849uc2p.dsql-fnh4.us-east-1.vpce.amazonaws.com
 # This should match your endpoint's fully-qualified domain name

# Construct the hostname used to generate the authentication token
export AUTH_HOSTNAME="$CLUSTERID.dsql.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $AUTH_HOSTNAME)

# Specify the amzn-cluster-id connection option
export PGOPTIONS="-c amzn-cluster-id=$CLUSTERID"
```

```
# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Risoluzione dei problemi relativi a AWS PrivateLink

Problemi e soluzioni comuni

La tabella seguente elenca i problemi e le soluzioni comuni relativi a AWS PrivateLink con Aurora DSQL.

Problema	Possibile causa	Soluzione
Timeout di connessione	Gruppo di sicurezza non configurato correttamente	Utilizza il sistema di analisi della reperibilità Amazon VPC per assicurarti che la configurazione della rete consenta il traffico sulla porta 5432.
Errore di risoluzione del DNS	DNS privato non abilitato	Verifica che l'endpoint Amazon VPC sia stato creato con DNS privato abilitato.
Errori di autenticazione	Credenziali errate o token scaduto	Genera un nuovo token di autenticazione e verificare il nome utente.
Il nome del servizio non è stato trovato	ID del cluster non corretto	Ricontrolla l'ID del cluster e Regione AWS quando recuperi il nome del servizio.

Risorse correlate

Per maggiori informazioni, consulta le seguenti risorse:

- Guida per l'utente di Amazon Aurora DSQL
- Documentazione di AWS PrivateLink
- Accedi ai servizi tramite AWSAWS PrivateLink

Analisi della configurazione e delle vulnerabilità in Amazon Aurora DSQL

AWSgestisce le attività di sicurezza di base come l'applicazione di patch al sistema operativo guest (OS) e al database, la configurazione del firewall e il disaster recovery. Queste procedure sono state riviste e certificate dalle terze parti appropriate. Per ulteriori dettagli, consulta le seguenti risorse :

- Modello di responsabilità condivisa
- Amazon Web Services: panoramica dei processi di sicurezza (whitepaper)

Prevenzione del confused deputy tra servizi

Il problema confused deputy è un problema di sicurezza in cui un'entità che non dispone dell'autorizzazione per eseguire un'azione può costringere un'entità maggiormente privilegiata a eseguire l'azione. Nel frattempoAWS, l'impersonificazione tra servizi può portare al confuso problema del vicesceriffo. La rappresentazione tra servizi può verificarsi quando un servizio (il servizio chiamante) effettua una chiamata a un altro servizio (il servizio chiamato). Il servizio chiamante può essere manipolato per utilizzare le proprie autorizzazioni e agire sulle risorse di un altro cliente, a cui normalmente non avrebbe accesso. Per evitare ciò, AWS fornisce strumenti per poterti a proteggere i tuoi dati per tutti i servizi con entità di servizio a cui è stato concesso l'accesso alle risorse del tuo account.

Consigliamo di utilizzare le chiavi di contesto delle condizioni globali `aws:SourceArn` e `aws:SourceAccount` nelle policy delle risorse per limitare le autorizzazioni con cui Amazon Aurora DSQL fornisce un altro servizio alla risorsa. Utilizzare `aws:SourceArn` se si desidera consentire l'associazione di una sola risorsa all'accesso tra servizi. Utilizzare `aws:SourceAccount` se si desidera consentire l'associazione di qualsiasi risorsa in tale account all'uso tra servizi.

Il modo più efficace per proteggersi dal problema "confused deputy" è quello di utilizzare la chiave di contesto della condizione globale `aws:SourceArn` con l'ARN completo della risorsa. Se non si conosce l'ARN completo della risorsa o si scelgono più risorse, utilizzare la chiave di contesto della condizione globale `aws:SourceArn` con caratteri jolly (*) per le parti sconosciute dell'ARN. Ad esempio, `arn:aws:dsql:*:123456789012:*`.

Se il valore `aws:SourceArn` non contiene l'ID account, ad esempio un ARN di un bucket Amazon S3, è necessario utilizzare entrambe le chiavi di contesto delle condizioni globali per limitare le autorizzazioni.

Il valore di `aws:SourceArn` deve essere ResourceDescription.

L'esempio seguente mostra il modo in cui puoi utilizzare le chiavi di contesto delle condizioni globali `aws:SourceArn` e `aws:SourceAccount` in Aurora DSQL per prevenire il problema confused deputy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "backup.amazonaws.com"
    },
    "Action": "dsql:GetCluster",
    "Resource": [
      "arn:aws:dsql:*:123456789012:cluster/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:backup:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Best practice di sicurezza di Aurora DSQL

Aurora DSQL fornisce una serie di funzionalità di sicurezza da valutare durante lo sviluppo e l'implementazione delle policy di sicurezza. Le seguenti best practice sono linee guida generali e non rappresentano una soluzione di sicurezza completa. Poiché queste best practice potrebbero non essere appropriate o sufficienti per l'ambiente, sono da considerare come considerazioni utili anziché prescrizioni.

Argomenti

Best practice relative alla sicurezza di rilevamento per Aurora DSQL

Oltre ai seguenti modi per utilizzare in modo sicuro Aurora DSQL, consulta Sicurezza AWS Well-Architected Tool in per scoprire come le tecnologie cloud migliorano la tua sicurezza.

CloudWatch Allarmi Amazon

Utilizzando Amazon CloudWatch alarms, controlli una singola metrica per un periodo di tempo specificato. Se la metrica supera una determinata soglia, viene inviata una notifica a un argomento o una policy di Amazon SNS. AWS Auto Scaling CloudWatch gli allarmi non richiamano azioni perché si trovano in uno stato particolare. È necessario invece cambiare lo stato e mantenerlo per un numero di periodi specificato.

Tagging delle risorse di Aurora DSQL per identificazione e automazione

Puoi assegnare metadati alle tue AWS risorse sotto forma di tag. Ogni tag è una semplice etichetta composta da una chiave definita dal cliente e un valore facoltativo che può semplificare la gestione, la ricerca e il filtro delle risorse.

Il tagging consente l'implementazione di gruppi controllati. Anche se non ci sono tipi di tag inerenti, è possibile suddividere le risorse in base a scopo, proprietario, ambiente o altri criteri. Di seguito vengono mostrati alcuni esempi:

- Sicurezza - Utilizzato per determinare requisiti quali la crittografia.
- Riservatezza - Un identificatore per il livello di riservatezza dei dati specifico supportato da una risorsa.
- Ambiente - Utilizzato per differenziare tra infrastruttura di sviluppo, test e produzione.

Puoi assegnare metadati alle tue AWS risorse sotto forma di tag. Ogni tag è una semplice etichetta composta da una chiave definita dal cliente e un valore facoltativo che può semplificare la gestione, la ricerca e il filtro delle risorse.

Il tagging consente l'implementazione di gruppi controllati. Anche se non ci sono tipi di tag inerenti, i tag consentono di suddividere le risorse in base a scopo, proprietario, ambiente o altri criteri. Di seguito vengono mostrati alcuni esempi.

- Sicurezza - Utilizzato per determinare requisiti quali la crittografia.

- Riservatezza - Un identificatore per il livello di riservatezza dei dati specifico supportato da una risorsa.
- Ambiente Utilizzato per differenziare tra infrastruttura di sviluppo, test e produzione.

Per ulteriori informazioni, consulta Best practice per l'AWSetichettatura delle risorse.

Best practice relative alla sicurezza preventiva per Aurora DSQL

Oltre ai seguenti modi per utilizzare in modo sicuro Aurora DSQL, consulta Sicurezza AWS Well-Architected Tool in per scoprire come le tecnologie cloud migliorano la tua sicurezza.

Utilizza i ruoli IAM per autenticare l'accesso ad Aurora DSQL.

Gli utenti, le applicazioni e gli altri utenti Servizi AWS che accedono ad Aurora DSQL devono includere AWS credenziali valide nell'API e nelle AWS richieste. AWS CLI Non è necessario archiviare AWS le credenziali direttamente nell'applicazione o nelle istanze. EC2 Si tratta di credenziali a lungo termine che non vengono ruotate automaticamente. L'eventuale compromissione di queste credenziali ha un impatto significativo sul business. Un ruolo IAM ti consente di ottenere chiavi di accesso temporanee da utilizzare per accedere a risorse Servizi AWS e risorse.

Per ulteriori informazioni, consulta Autenticazione e autorizzazione per Aurora DSQL.

Utilizza le policy IAM per l'autorizzazione di base di Aurora DSQL.

Quando si concedono le autorizzazioni, si decide gli utenti che le riceveranno, quali API di Aurora DSQL ottengono le autorizzazioni e le operazioni specifiche da consentire su tali risorse. L'implementazione del privilegio minimo è fondamentale per ridurre i rischi di sicurezza e l'impatto che può risultare da errori o intenzioni dannose.

Collega policy di autorizzazione ai ruoli IAM e concedi autorizzazioni per eseguire operazioni su risorse Aurora DSQL. Sono disponibili anche limiti delle autorizzazioni per le entità IAM, che consentono di impostare le autorizzazioni massime che una policy basata sull'identità può concedere a un'entità IAM.

Analogamente alle best practice per gli utenti root Account AWS, non utilizzare il `admin` ruolo in Aurora DSQL per eseguire operazioni quotidiane. Consigliamo invece di creare ruoli del database personalizzati per gestire e connettersi al cluster. Per maggiori informazioni, consulta Accesso ad Aurora DSQL e Informazioni sull' autenticazione e l'autorizzazione per Aurora DSQL.

Utilizzo di **`verify-full`** in ambienti di produzione.

Questa impostazione verifica che il certificato del server sia firmato da un'autorità di certificazione affidabile e che il nome host del server corrisponda al certificato.

Aggiornamento del client PostgreSQL

Aggiorna regolarmente il tuo client PostgreSQL alla versione più recente per beneficiare dei miglioramenti della sicurezza. Si consiglia di utilizzare PostgreSQL versione 17.

Applicazione di tag alle risorse in Aurora DSQL

In AWS, i tag sono coppie chiave-valore definite dall'utente che vengono definite e associate a risorse Aurora DSQL come i cluster. I tag sono opzionali. Se viene fornita una chiave, il valore è facoltativo.

È possibile utilizzare la Console di gestione AWS, la AWS CLI o gli SDK AWS per aggiungere, elencare ed eliminare tag sui cluster Aurora DSQL. È possibile aggiungere tag durante e dopo la creazione del cluster utilizzando la console AWS. Per etichettare un cluster dopo la creazione, la AWS CLI usa l'operazione `TagResource`.

Applicazione di tag ai cluster con un nome

Aurora DSQL crea cluster con un identificatore unico a livello globale assegnato come nome della risorsa Amazon (ARN). Se si desidera assegnare un nome intuitivo al cluster, raccomandiamo di utilizzare un Tag.

Se si crea una console con la console di Aurora DSQL, Aurora DSQL crea automaticamente un tag. Questo tag è costituito da una chiave Name e un valore generato automaticamente che rappresenta il nome del cluster. Questo valore è configurabile, quindi è possibile assegnare al cluster un nome più intuitivo. Se a un cluster è stato applicato un tag Name con un valore associato, è possibile visualizzare il valore in tutta la console Aurora DSQL.

Requisiti per il tagging

I tag hanno i requisiti seguenti:

- Alle chiavi non può essere anteposto il prefisso `aws:`.
- Le chiavi devono essere univoche per un set di tag.
- Una chiave deve essere costituita da un numero di caratteri compreso tra 1 e 128.
- Un valore deve essere costituito da un numero di caratteri compreso tra 0 e 256.
- Non è necessario che i valori siano univoci per un set di tag.
- I caratteri consentiti per le chiavi e i valori sono lettere, cifre, spazi e uno qualsiasi dei simboli seguenti: _ . : / = + - @.
- Per chiavi e valori viene fatta distinzione tra maiuscole e minuscole.

Note di utilizzo dell'applicazione di tag

Quando si usano i tag in Aurora DSQL, tenere presenti le seguenti considerazioni.

- Quando si usano la AWS CLI o le operazioni API di Aurora DSQL, è necessario fornire il nome della risorsa Amazon (ARN) della risorsa Aurora DSQL che si intende utilizzare. Per maggiori informazioni, consultare Formato del nome della risorsa Amazon (ARN) per le risorse di Aurora DSQL.
- Ogni risorsa dispone di un set di tag, ovvero una raccolta di uno o più tag ad essa assegnati.
- Ogni risorsa può avere fino a 50 tag per set di tag.
- Se si elimina una risorsa, vengono eliminati anche i tag associati.
- È possibile aggiungere tag alla creazione della risorsa, è possibile visualizzare e modificare i tag utilizzando le seguenti operazioni API: `TagResource`, `UntagResource` e `ListTagsForResource`.
- È possibile utilizzare i tag con le policy IAM. È possibili utilizzarli per gestire l'accesso ai cluster Aurora DSQL e per controllare le operazioni che è possibile applicare alle specifiche risorse. Per maggiori informazioni, consultare Controllo dell'accesso alle risorse di AWS utilizzando i tag.
- È possibile utilizzare i tag per diverse altre attività in tutto AWS. Per maggiori informazioni, consultare Strategie comuni di applicazione di tag.

Considerazioni sull'utilizzo di Amazon Aurora DSQL

Quando si lavora con Amazon Aurora DSQL è opportuno prendere in considerazione i seguenti comportamenti. Per maggiori informazioni sulla compatibilità e il supporto di PostgreSQL, consultare Compatibilità delle funzionalità SQL in Aurora DSQL. Per quote e limiti, consultare Quote di cluster e limiti del database in Amazon Aurora DSQL.

- Aurora DSQL non completa le transazioni `COUNT(*)` prima del timeout della transazione per tabelle di grandi dimensioni. Per recuperare il conteggio delle righe della tabella dal catalogo di sistema, consultare Utilizzo delle tabelle e dei comandi di sistema in Aurora DSQL.
- I driver che richiamano `PG_PREPARED_STATEMENTS` potrebbero fornire una visualizzazione incoerente delle istruzioni preparate memorizzate nella cache per il cluster. Potrebbe essere visualizzato un numero di istruzioni preparate per connessione superiore al previsto per lo stesso cluster e lo stesso ruolo IAM. Aurora DSQL non conserva i nomi delle istruzioni preparati dall'utente.
- In rari scenari di compromissione dei cluster multi-Regione, il ripristino della disponibilità del commit delle transazioni potrebbe richiedere più tempo del previsto. In generale, le operazioni automatizzate di ripristino dei cluster possono causare errori transitori nel controllo della concorrenza o nella connessione. Nella maggior parte dei casi, gli effetti saranno visibili solo per una percentuale del carico di lavoro. Quando si riscontrano tali errori transitori, riprova la transazione o riconnettere il client.
- Alcuni client SQL, come Datagrip, effettuano chiamate estese ai metadati di sistema per compilare le informazioni dello schema. Aurora DSQL non supporta tutte queste informazioni e restituisce degli errori. Questo problema non influisce sulla funzionalità delle query SQL, ma potrebbe influire sulla visualizzazione dello schema.
- Il ruolo di amministratore dispone di una serie di autorizzazioni relative alle attività di gestione del database. Per impostazione predefinita, queste autorizzazioni non si estendono agli oggetti creati da altri utenti. Il ruolo di amministratore non può concedere o revocare le autorizzazioni su questi oggetti creati dall'utente ad altri utenti. L'utente amministratore può concedersi qualsiasi altro ruolo per ottenere le autorizzazioni necessarie su questi oggetti.

Quote di cluster e limiti del database in Amazon Aurora DSQL

Le sezioni seguenti descrivono le quote del cluster e i limiti del database per Aurora DSQL.

Quote del cluster

Hai Account AWS le seguenti quote di cluster in Aurora DSQL. Per richiedere un aumento delle quote di servizio per i cluster a regione singola e multiarea all'interno di uno specifico Regione AWS, utilizza la pagina della console Service Quotas. Per altri aumenti delle quote, contatta. Supporto AWS

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL
Numero massimo di cluster a regione singola per Account AWS	20 cluster	Sì	Codice di errore API `ServiceQuotaExceededExcep`
Numero massimo di cluster multiregione per Account AWS	5 cluster	Sì	Codice di errore API `ServiceQuotaExceededExcep`
Spazio di archiviazione massimo per cluster	Limite predefinito di 10 TiB, fino a 256 TiB con aumento del limite approvato	Sì	`DISK_FULL(53100)`

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL
Numero massimo di connessioni per cluster	10.000 connessioni	Sì	`TOO_MANY_CONNECTIONS(53300)`
Velocità massima di connessione per cluster	100 connessioni al secondo	No	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacità di espansione massima delle connessioni per cluster	1.000 connessioni	No	Nessun codice di errore
Numero massimo di processi di ripristino simultanei	4	No	Nessun codice di errore
Frequenza di ricarica della connessione	100 connessioni al secondo	No	Nessun codice di errore

Limiti del database in Aurora DSQL

La tabella seguente descrive i limiti del database in Aurora DSQL.

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL	Messaggio di errore
Dimensione massima combinata delle colonne utilizzate in una chiave primaria	1 KiB	No	54000	ERROR: key size too large
Dimensione massima combinata delle colonne in un indice secondario	1 KiB	No	54000	ERROR: key size too large
Dimensione massima di una riga in una tabella	2 MiB	No	54000	ERROR: maximum row size excee
Dimensione massima di una colonna che non fa parte di un indice	1 MiB	No	54000	ERROR: maximum column size ex
Numero massimo di colonne in una chiave primaria o in un indice secondario	8	No	54011	ERROR: more than 8 column key are not supported

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL	Messaggio di errore
Numero massimo di colonne in una tabella	255	No	`54011`	`ERROR: tables can have at mos`
Numero massimo di indici in una tabella	24	No	`54000`	`ERROR: more than 24 indexes p allowed`
Dimensione massima di tutti i dati modificati in una transazione di scrittura	10 MiB	No	`54000`	`ERROR: transaction size limit DETAIL: Current transaction s 10mb`
Numero massimo di righe di tabelle e indici che possono essere modificate in un blocco di transazione	3.000 righe per transazione. Consulta Considerazioni su Aurora DSQL rispetto alla compatibilità con PostgreSQL.	No	`54000`	`ERROR: transaction row limit`

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL	Messaggio di errore
Quantità massima di memoria di base utilizzabile da un'operazione di interrogazione	128 MiB per transazione	No	53200	ERROR: query requires too muc out of memory.
Numero massimo di schemi definiti in un database	10	No	54000	ERROR: more than 10 schemas n
Numero massimo di tabelle in un database	1.000 tabelle	No	54000	ERROR: creating more than 100 allowed
Numero massimo di database in un cluster	1	No	Nessun codice di errore	ERROR: unsupported statement
Tempo massimo di una transazione	5 minuti	No	54000	ERROR: transaction age limit exceeded
Durata massima di una connessione	60 minuti	No	Nessun codice di errore	Nessun messaggio di errore

Description	Limite predefinito	Configurabile?	Codice di errore di Aurora DSQL	Messaggio di errore
Numero massimo di viste un database	5.000	No	`54000`	`ERROR: creating more than 500 allowed`
Dimensione massima della definizione di una vista	2 MiB	No	`54000`	`ERROR: view definition too la`

Per i limiti dei tipi di dati specifici di Aurora DSQL, consulta Tipi di dati supportati in Aurora DSQL.

Guida di riferimento alle API di Aurora DSQL

Oltre alla Console di gestione AWS e alla AWS Command Line Interface (AWS CLI), Aurora DSQL fornisce anche un'interfaccia API. È possibile utilizzare le operazioni API per gestire le risorse in Aurora DSQL.

Per un elenco alfabetico delle operazioni API, consultare Operazioni.

Per un elenco alfabetico dei tipi di dati, consulta la pagina Tipi di dati.

Per un elenco di parametri di query comuni, consulta la pagina Parametri Comuni.

Per le descrizioni dei codici di errore, consultare Errori comuni.

Per maggiori informazioni sulla AWS CLI, consultare la guida di riferimento della AWS Command Line Interface per Aurora DSQL.

Risoluzione dei problemi in Aurora DSQL

> **Note**
>
> Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo di Aurora DSQL. Se riscontri un problema non elencato qui di seguito, contatta il servizio di assistenza AWS

Argomenti

Risoluzione dei problemi legati agli errori di connessione

errore: codice di errore SSL non riconosciuto: 6 o impossibile accettare la connessione, sni non è stato ricevuto

Potresti utilizzare una versione di psql precedente alla versione 14, che non supporta Server Name Indication (SNI). Il supporto di SNI è necessario per la connessione ad Aurora DSQL.

È possibile verificare la versione del client con il comando `psql --version`.

errore: NetworkUnreachable

Un `NetworkUnreachable` errore durante i tentativi di connessione potrebbe indicare che il client non supporta IPv6 le connessioni, anziché segnalare un vero problema di rete. Questo errore si verifica in genere IPv4 solo sulle istanze a causa del modo in cui i client PostgreSQL gestiscono le connessioni dual-stack. Quando un server supporta la modalità dual-stack, questi client risolvono innanzitutto i nomi host in entrambi gli indirizzi. IPv4 IPv6 Tentano prima una IPv4 connessione, poi provano IPv6 se la connessione iniziale fallisce. Se il tuo sistema non lo supporta IPv6, vedrai un `NetworkUnreachable` errore generico invece di un chiaro messaggio «IPv6 non supportato».

Risoluzione dei problemi legati agli errori di autenticazione

Autenticazione IAM non riuscita per l'utente "..."

Quando si genera un token di autenticazione Aurora DSQL IAM, la durata massima che è possibile impostare è di 1 settimana. Dopo una settimana, non è più possibile autenticarsi con quel token.

Inoltre, Aurora DSQL rifiuta la richiesta di connessione se il ruolo assunto è scaduto. Ad esempio, se si prova a connettersi con un ruolo IAM temporaneo, Aurora DSQL rifiuterà la richiesta di connessione anche se il token di autenticazione non è scaduto.

Per maggiori informazioni su come IAM funziona con Aurora DSQL, consulta Comprendere l'autenticazione e l'autorizzazione per Aurora DSQL e AWS Identity and Access Management in Aurora DSQL.

Si è verificato un errore (InvalidAccessKeyId) durante la chiamata dell' GetObjectoperazione: l'ID della chiave di AWS accesso che hai fornito non esiste nei nostri archivi

IAM ha rifiutato la richiesta. Per informazioni, consulta Perché le richieste sono firmate.

Il ruolo IAM <ruolo> non esiste

Aurora DSQL non è riuscita a trovare il ruolo IAM. Per maggiori informazioni, consulta Ruoli IAM.

Il ruolo IAM deve assumere la forma di un ARN IAM

Per ulteriori informazioni, consulta IAM Identifiers - IAM ARNs.

Risoluzione dei problemi legati agli errori di autorizzazione

Ruolo <ruolo> non supportato

Aurora DSQL non supporta l'operazione `GRANT`. Consulta Sottoinsiemi di comandi PostgreSQL supportati in Aurora DSQL.

Impossibile stabilire un rapporto di fiducia con il ruolo <ruolo>

Aurora DSQL non supporta l'operazione `GRANT`. Consulta Sottoinsiemi di comandi PostgreSQL supportati in Aurora DSQL.

Il ruolo <ruolo> non esiste

Aurora DSQL non è riuscita a trovare l'utente del database specificato. Consulta Autorizzare i ruoli di database personalizzati per la connessione a un cluster.

ERRORE: autorizzazione negata nella concessione della fiducia IAM con il ruolo <ruolo>

Per concedere l'accesso a un ruolo del database, bisogna essere connessi al cluster con il ruolo di amministratore. Per maggiori informazioni, consulta Autorizzare i ruoli del database all'utilizzo di SQL in un database.

ERRORE: il ruolo <ruolo> deve possedere l'attributo LOGIN

Tutti i ruoli del database creati devono possedere l'autorizzazione `LOGIN`.

Per risolvere questo errore, assicurarsi di aver creato il ruolo PostgreSQL con l'autorizzazione `LOGIN`. Per maggiori informazioni, consulta CREATE ROLE e ALTER ROLE nella documentazione di PostgreSQL.

ERRORE: il ruolo <ruolo> non può essere eliminato perché alcuni oggetti dipendono da esso

Aurora DSQL restituisce un errore se si elimina un ruolo di database con una relazione IAM finché non si revoca la relazione utilizzando `AWS IAM REVOKE`. Per maggiori informazioni, consulta Revoca dell'autorizzazione.

Risoluzione degli errori SQL

Errore: Non supportato

Aurora DSQL non supporta tutti i dialetti basati su PostgreSQL. Per informazioni su ciò che è supportato, consulta Funzionalità PostgreSQL supportate in Aurora DSQL.

Errore: utilizzare invece **`CREATE INDEX ASYNC`**

Per creare un indice su una tabella con righe esistenti, è necessario utilizzare il comando `CREATE INDEX ASYNC`. Per maggiori informazioni, consulta Creazione di indici in modo asincrono in Aurora DSQL.

Risoluzione degli errori OCC

OC000 "ERRORE: la mutazione è in conflitto con un'altra transazione, riprovare se necessario"

Questa transazione ha tentato di modificare le stesse tuple di un'altra transazione simultanea. Ciò indica una contesa sulle tuple modificate. Per ulteriori informazioni, consulta il controllo della concorrenza in Aurora DSQL

OC001 "ERRORE: lo schema è stato aggiornato da un'altra transazione, riprovare se necessario"

La sessione PostgreSQL aveva una copia cache del catalogo dello schema. La copia memorizzata nella cache era valida all'istante del caricamento. Chiamiamo l'istante T1 e la versione V1.

Un'altra transazione aggiorna il catalogo all'istante T2. Chiamiamo questa versione V2.

Quando la sessione originale tenta di leggere dalla memoria al momento T2, utilizza ancora la versione del catalogo V1. Il livello di archiviazione di Aurora DSQL rifiuta la richiesta perché l'ultima versione del catalogo in T2 è la V2.

Quando si riprovi all'istante T3 dalla sessione originale, Aurora DSQL aggiorna la cache del catalogo. La transazione in T3 utilizza il catalogo V2. Aurora DSQL completa la transazione a condizione che non siano state apportate altre modifiche al catalogo dall'istante T2.

Risoluzione dei problemi di connessione SSL/TLS

Errore SSL: verifica del certificato non riuscita

Questo errore indica che il client non è in grado di verificare il certificato del server. Verifica che:

1. Il certificato Amazon Root CA 1 sia installato correttamente. Consulta Configurazione dei SSL/TLS certificati per le connessioni Aurora DSQL per le istruzioni su come convalidare e installare questo certificato.
2. La variabile di ambiente `PGSSLROOTCERT` punti al file di certificato corretto.
3. Il file del certificato disponga delle autorizzazioni corrette.

Codice di errore SSL non riconosciuto: 6

Questo errore si verifica con i client PostgreSQL precedenti alla versione 14. Aggiornare il client PostgreSQL alla versione 17 per risolvere questo problema.

Errore SSL: schema non registrato (Windows)

Si tratta di un problema noto del client Windows psql quando si utilizzano i certificati di sistema. Utilizza il metodo del file di certificato scaricato descritto nelle istruzioni di Connessione da Windows.

Fornire feedback su Amazon Aurora DSQL

Se riscontri funzionalità fondamentali per la migrazione ma che attualmente non sono supportate in Aurora DSQL, AWS fornisce diversi canali per il feedback:

Canali di feedback

Server Discord Aurora DSQL

: Unisciti al server Discord Aurora DSQL per connetterti con il team e la community AWS. Condividi le richieste di funzionalità, discuti le sfide della migrazione e ottieni feedback in tempo reale.

AWS Support

: Se disponi di un piano AWS Support, crea un caso di supporto per discutere dei tuoi requisiti specifici e delle tue esigenze temporali.

AWS re:POST

: Usa AWS re:POST per porre domande e condividere feedback con la community e gli esperti AWS.

Richieste di funzionalità efficaci

Quando richiedi funzionalità, fornisci:

- Descrizione del caso d'uso: spiega cosa stai cercando di ottenere e perché
- Soluzione attuale: descrivi tutte le alternative che hai provato
- Impatto sull'azienda: spiegate in che modo la funzionalità mancante influisce sulla tempistica della migrazione o sulla funzionalità dell'applicazione
- Livello di priorità: indica se ciò sta bloccando la migrazione o rappresenterebbe un miglioramento nice-to-have

Cronologia dei documenti per la Guida per l'utente di Amazon Aurora DSQL

La tabella seguente descrive i rilasci della documentazione per Aurora DSQL.

Modifica	Descrizione	Data
Supporto di policy basato sulle risorse per Amazon Aurora DSQL	È stato aggiunto il supporto di policy basate sulle risorse (RBP) con nuove autorizzazioni:,, e. `PutClusterPolicy` `GetClusterPolicy` `DeleteClusterPolicy` Queste autorizzazioni consentono di gestire le policy in linea collegate ai cluster Aurora DSQL per un controllo granulare degli accessi. Politiche gestite aggiornate AmazonAurora DSQLFull Accesso e inclusione delle funzionalità RBP. AmazonAurora DSQLRead OnlyAccess AmazonAurora DSQLConsole FullAccess Per ulteriori informazioni, consulta le politiche AWS gestite per Amazon Aurora DSQL.	15 ottobre 2025
Wrapper JDBC per Aurora DSQL	È stata aggiunta documentazione per il wrapper JDBC per Aurora DSQL, un wrapper PgJDBC che integra l'autenticazione IAM per la connessione di applicazioni Java ai	2 settembre 2025

	cluster Amazon Aurora DSQL. Per maggiori informazioni, consultare Utilizzo del wrapper JDBC per Aurora DSQL.	
AWS aggiornamenti delle politiche gestite per l'integrazione AWS FIS	`AmazonAuroraDSQLConsoleFullAccess` Politiche `AmazonAuroraDSQLFullAccess` e politiche aggiornate per supportare AWS Fault Injection Service l'integrazione con Aurora DSQL. Ciò consente di inserire errori in cluster Aurora DSQL a Regione singola e multi-Regione per testare la tolleranza ai guasti delle applicazioni. Per maggiori informazioni su queste politiche, consultare Aggiornamenti delle policy gestite di AWS.	19 agosto 2025
Disponibilità generale (GA) di Amazon Aurora DSQL	Amazon Aurora DSQL è ora disponibile a livello generale con supporto aggiuntivo per il CloudWatch monitoraggio, funzionalità avanzate di protezione dei dati e integrazione. AWS Backup Per ulteriori informazioni, consulta Monitoraggio di Aurora DSQL con, CloudWatch Backup e ripristino per Amazon Aurora DSQL e Crittografia dei dati per Amazon Aurora DSQL.	27 maggio 2025

AmazonAuroraDSQLFullAccedi all'aggiornamento	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio delle attività. Aggiunge inoltre la possibilità di utilizzare chiavi KMS gestite dal cliente per la crittografia dei cluster. Per ulteriori informazioni, vedere AmazonAuroraDSQLFullAccesso e utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	21 maggio 2025
AmazonAuroraDSQLConsoleFullAccess update	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL tramite la AWS Console Home. Ciò include l'avvio, l'arresto e il monitoraggio delle attività. Supporta anche l'utilizzo di chiavi KMS gestite dal cliente per la crittografia dei cluster e l'avvio di AWS CloudShell. Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccessUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	21 maggio 2025

AmazonAuroraDSQLReadOnlyAccess aggiornare	Include la possibilità di determinare il nome corretto del servizio endpoint VPC durante la connessione ai cluster Aurora DSQL tramite AWS PrivateLink Aurora DSQL crea endpoint unici per cella, quindi questa API aiuta a identificare l'endpoint corretto per il cluster ed evitare errori di connessione. Per ulteriori informazioni, vedere AmazonAuroraDSQLReadOnlyAccessUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

AmazonAuroraDSQLFullAccedi all'aggiornamento	La politica aggiunge quattro nuove autorizzazioni per creare e gestire cluster di database su più livelli Regioni AWS:`PutMultiRegionProperties` , `PutWitnessRegion` `AddPeerCluster` , e. `RemovePeerCluster` Queste autorizzazioni includono controlli a livello di risorsa e chiavi di condizione in modo da poter controllare quali cluster possono essere modificati dagli utenti. La policy aggiunge anche l'`GetVpcEndpointServiceName` autorizzazione per aiutarti a connetterti ai tuoi cluster Aurora DSQL tramite. AWS PrivateLink Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccessUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

AmazonAuroraDSQLConsoleFullAccess update	Aggiunge nuove autorizzazioni ad Aurora DSQL per supportare la gestione di cluster multi-Regione e la connessione agli endpoint VPC. Le nuove autorizzazioni includono: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` , `RemovePeerCluster` e `GetVpcEndpointServiceName` . Vedi AmazonAuroraDSQLConsoleFullAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025
AuroraDsqlServiceLinkedRolePolicy update	Aggiunge la possibilità di pubblicare metriche nei namespace `AWS/AuroraDSQL` e `AWS/Usage` `CloudWatch` della policy. Ciò consente al servizio o al ruolo associato di inviare dati più completi sull'utilizzo e sulle prestazioni all'ambiente. CloudWatch Per ulteriori informazioni, vedere AuroraDsqlServiceLinkedRolePolicyUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	8 maggio 2025

AWS PrivateLink per Amazon Aurora DSQL	Aurora DSQL ora supporta. AWS PrivateLink Con AWS PrivateLink, puoi semplificare la connettività di rete privata tra cloud privati virtuali (VPCs), Aurora DSQL e i data center locali utilizzando l'interfaccia Amazon VPC, endpoint e indirizzi IP privati. Per maggiori informazioni, consultare Gestione e connessione ai cluster di Amazon Aurora DSQL utilizzando AWS PrivateLink.	8 maggio 2025
Versione iniziale	Versione iniziale della Guida per l'utente di Amazon Aurora DSQL.	3 dicembre 2024

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.