aws

Guida per l'utente

Amazon Aurora DSQL

Amazon Aurora DSQL: Guida per l'utente

Table of Contents

Cos'è Amazon Aurora DSQL?

Amazon Aurora DSQL è un servizio di database relazionale distribuito e senza server ottimizzato per 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 usare Aurora DSQL

Aurora DSQL è ottimizzata 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 ridimensiona l'elaborazione, l'I/O e lo storage in base al carico di lavoro. Poiché non avete server da fornire o gestire, non dovete preoccuparvi dei tempi di inattività per manutenzione legati al provisioning, all'applicazione di patch o agli aggiornamenti dell'infrastruttura.

Aurora DSQL ti aiuta a creare e mantenere applicazioni aziendali sempre disponibili su qualsiasi scala. Il design serverless active-active automatizza il ripristino dagli errori, quindi non devi preoccuparti del tradizionale failover del database. Le vostre applicazioni traggono vantaggio dalla disponibilità multi-AZ e multiregione e non dovete preoccuparvi dell'eventuale coerenza o della mancanza di dati relativi ai failover.

Caratteristiche principali di Aurora DSQL

Le seguenti funzionalità chiave consentono di creare un database distribuito senza server per supportare le applicazioni ad alta disponibilità:

Architettura distribuita

Aurora DSQL è composto dai seguenti componenti multi-tenant:

- Relè e connettività
- Elaborazione e database

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

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

Cluster a regione singola e multiregione

I cluster Aurora DSQL offrono i seguenti vantaggi:

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

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

I cluster peer multiregione offrono la stessa resilienza e connettività dei cluster a regione singola. Tuttavia, migliorano la disponibilità offrendo due endpoint regionali, uno in ogni regione del cluster peer-ered. Entrambi gli endpoint di un cluster peer 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. Puoi connetterti ad Aurora DSQL con driver e strumenti PostgreSQL familiari, 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 ulteriori informazioni sulle funzionalità SQL 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, il servizio 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

Supportati ed endpoint Regioni AWS

Nome Regione	Regione	Endpoint	Protocollo
US East (N. Virginia)	us-east-1	dsql.us-east-1.api.aws	HTTPS
Stati Uniti orientali (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
US West (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS
Europa (Irlanda)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europe (London)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europe (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
Asia Pacifico (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Asia Pacifico (Seul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asia Pacifico (Osaka-Locale)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS

Disponibilità di cluster multiregionali 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 multiregionale.

Regioni degli Stati Uniti

- Stati Uniti orientali (Virginia settentrionale)
- Stati Uniti orientali (Ohio)
- US West (Oregon)

Regioni dell'Asia Pacifico

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

Regioni europee

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

Limitazioni importanti

I cluster multiregione 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 attualmente non supporta cluster multiregionali intercontinentali.

Prezzi di Aurora DSQL

Per informazioni sui costi, consulta i prezzi di Aurora DSQL.

Fasi successive

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

- Guida introduttiva ad Aurora DSQL
- Compatibilità delle funzionalità SQL in Aurora DSQL
- Accesso ad Aurora SQL
- Aurora DSQL e PostgreSQL

Guida introduttiva ad Aurora DSQL

Amazon Aurora DSQL è un database relazionale distribuito e senza server 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 AWS Management Console e interagirai con il tuo database utilizzando l'utilità. `psql` Alla fine, avrai un cluster Aurora DSQL funzionante configurato e pronto all'uso per carichi di lavoro di test o produzione.

Argomenti

Prerequisiti

Prima di iniziare a utilizzare Aurora DSQL, assicurati di soddisfare i seguenti prerequisiti:

- La tua identità IAM deve disporre dell'autorizzazione per accedere a. AWS Management Console
- La tua identità IAM deve soddisfare uno dei seguenti criteri:
  - Accedi per eseguire qualsiasi azione su qualsiasi risorsa del tuo Account AWS
  - L'autorizzazione IAM `iam:CreateServiceLinkedRole` e la possibilità di accedere all'azione politica IAM `dsql:*`
- Se lo usi AWS CLI in un ambiente simile a Unix, assicurati che siano installate la versione 3.8+ e la versione 14+ di Python. `psql` Per verificare le versioni dell'applicazione, esegui i seguenti comandi.

```
python3 --version
psql --version
```

Se usi Python AWS CLI in un ambiente diverso, assicurati di configurare manualmente la versione 3.8+ e la versione 14+ di Python. `psql`

- Se intendi accedere ad Aurora DSQL utilizzando, le versioni 3.8+ e 14+ di AWS CloudShell Python vengono fornite senza alcuna configurazione `psql` aggiuntiva. Per ulteriori informazioni su, consulta What is? AWS CloudShellAWS CloudShell .
- Se intendi accedere ad Aurora DSQL utilizzando una GUI, usa o. DBeaver JetBrains DataGrip Per ulteriori informazioni, consultare Accesso ad Aurora DSQL con DBeaver e Accesso ad Aurora DSQL con JetBrains DataGrip.

Accesso ad Aurora SQL

È possibile accedere ad Aurora DSQL tramite le seguenti tecniche. Per informazioni su come utilizzare la CLI, e APIs SDKs, consulta. Accesso ad Aurora SQL

Argomenti

- Accesso ad Aurora DSQL tramite AWS Management Console
- Accesso ad Aurora DSQL tramite client SQL
- Utilizzo del protocollo PostgreSQL con Aurora SQL

Accesso ad Aurora DSQL tramite AWS Management Console

È possibile accedere a AWS Management Console per Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql

Accesso ad Aurora DSQL tramite client SQL

Aurora DSQL utilizza il protocollo PostgreSQL. Usa il tuo client interattivo preferito fornendo un token di autenticazione IAM firmato come password per la connessione al cluster. Un token di autenticazione è una stringa di caratteri univoca che Aurora DSQL genera dinamicamente utilizzando AWS la versione 4 di Signature.

Aurora DSQL utilizza il token solo per l'autenticazione. Il token non influisce sulla connessione dopo che è stata stabilita. Se si tenta di riconnettersi utilizzando un token scaduto, la richiesta di connessione viene rifiutata. Per ulteriori informazioni, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.

Argomenti

Accesso ad Aurora DSQL con psql (terminale interattivo PostgreSQL)

L'`psql`utilità è un front-end basato su terminali per PostgreSQL. Consente di digitare le query in modo interattivo, inviarle a PostgreSQL e visualizzare i risultati delle query. Per migliorare i tempi di risposta alle query, usa il client PostgreSQL versione 17.

Scarica il programma di installazione del tuo sistema operativo dalla pagina dei download di PostgreSQL. Per ulteriori informazioni su`psql`, consulta https://www.postgresql. org/docs/current/app-psql.htm.

Se lo hai già AWS CLI installato, usa il seguente esempio per connetterti al tuo cluster. Puoi utilizzare AWS CloudShell, fornito con la versione `psql` preinstallata, oppure puoi installarlo `psql` direttamente.

```
# 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
```

Accesso ad Aurora DSQL con DBeaver

DBeaver è uno strumento di database open source basato su una GUI. Puoi usarlo per connetterti e gestire il tuo database. Per effettuare il download DBeaver, consulta la pagina di download sul sito Web della DBeaver Community. I passaggi seguenti spiegano come connettersi al cluster utilizzando DBeaver.

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: immettere `postgres`

     Autenticazione: scegli `Database Native`

     Nome utente: inserisci `admin`

     Password: genera un token di autenticazione. Copia il token generato e usalo come password.
4. Ignora qualsiasi avviso e incolla il token di autenticazione nel campo DBeaverPassword.

> **Note**
>
> È necessario impostare la modalità SSL nelle connessioni client. Aurora DSQL supporta. `SSLMODE=require` Aurora DSQL applica la comunicazione SSL sul lato server e rifiuta le connessioni non SSL.

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 a un database, a causa della sua architettura

unica. Sebbene accessibili, queste schermate non forniscono informazioni affidabili sullo stato o sullo stato del database.

Scadenza delle credenziali di autenticazione per DBeaver

Le sessioni stabilite rimangono autenticate per un massimo di 1 ora o fino alla DBeaver disconnessione o al timeout. Per stabilire nuove connessioni, fornisci un token di autenticazione valido nel campo Password delle impostazioni di connessione. Il tentativo di aprire una nuova sessione (ad esempio, per elencare nuove tabelle o una nuova console SQL) impone un nuovo tentativo di autenticazione. Se il token di autenticazione configurato nelle impostazioni di connessione non è più valido, la nuova sessione ha esito negativo e DBeaver invalida tutte le sessioni aperte in precedenza. Tienilo a mente quando scegli la durata del token di autenticazione IAM con l'`expires-in`opzione.

Accesso ad Aurora DSQL con JetBrains DataGrip

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 effettuare il download DataGrip, vai alla pagina di download sul JetBrains sito Web.

Per configurare una nuova connessione Aurora DSQL in JetBrains DataGrip

1. Scegli Nuova origine dati e scegli PostgreSQL.
2. Nella Sources/General scheda Dati, 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: scegli. `User & Password`

     Nome utente: immettere`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 del generatore di token e incollalo nel campo della password.

> **Note**
>
> È necessario impostare la modalità SSL nelle connessioni client. Aurora DSQL supporta. `PGSSLMODE=require` Aurora DSQL applica la comunicazione SSL sul lato server e rifiuterà le connessioni non SSL.

4. È necessario essere connessi al cluster e iniziare a eseguire istruzioni SQL:

> **Important**
>
> Alcune viste fornite dai DataGrip database PostgreSQL (come Sessions) non si applicano a un database a causa della sua architettura unica. Sebbene accessibili, queste schermate non forniscono informazioni affidabili sulle sessioni effettive connesse al database.

Scadenza delle credenziali di autenticazione

Le sessioni stabilite rimangono autenticate per un massimo di 1 ora o fino a quando non si verifica una disconnessione esplicita o un timeout lato client. Se è necessario stabilire nuove connessioni, è necessario generare e fornire un nuovo token di autenticazione nel campo Password delle proprietà dell'origine dei dati. Il tentativo di aprire una nuova sessione (ad esempio, per elencare nuove tabelle o una nuova console SQL) impone un nuovo tentativo di autenticazione. Se il token di autenticazione configurato nelle impostazioni di connessione non è più valido, la nuova sessione avrà esito negativo e tutte le sessioni aperte in precedenza non saranno più valide.

Utilizzo del protocollo PostgreSQL con Aurora SQL

PostgreSQL utilizza un protocollo basato su messaggi per la comunicazione tra client e server. Il protocollo è supportato più e anche tramite TCP/IP socket di dominio UNIX. La tabella seguente mostra come Aurora DSQL supporta il protocollo PostgreSQL.

PostgreSQL	Aurora SQL	Note
Ruolo (noto anche come utente o gruppo)	Ruolo di database	Aurora DSQL crea un ruolo chiamato per te. `admin` Quando crei ruoli di database

PostgreSQL	Aurora SQL	Note
		personalizzati, devi utilizzare il `admin` ruolo per associarli ai ruoli IAM per l'autenticazione durante la connessione al cluster. Per ulteriori informazioni, consulta Configurare i ruoli di database personalizzati.
Host (noto anche come hostname o hostspec)	Endpoint del cluster	I cluster Aurora DSQL Single-Region forniscono un unico endpoint gestito e reindirizzano automaticamente il traffico in caso di indisponibilità all'interno della regione.
Porta	N/A: usa l'impostazione predefinita `5432`	Questa è l'impostazione predefinita di PostgreSQL.
Database (dbname)	uso `postgres`	Aurora DSQL crea questo database per te quando crei il cluster.
Modalità SSL	SSL è sempre abilitato sul lato server	In Aurora DSQL, 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.

Fase 1: Creare un cluster Aurora DSQL a regione singola

L'unità base di Aurora DSQL è il cluster, dove vengono archiviati i dati. In questa attività, si crea un cluster in un'unica soluzione. Regione AWS

Per creare un cluster a regione singola in Aurora DSQL

1. Accedi a AWS Management Console e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. Scegli Crea cluster e poi Single-Region.
3. (Facoltativo) Nelle impostazioni del cluster, seleziona una delle seguenti opzioni:
   - Seleziona Personalizza le impostazioni di crittografia (avanzate) per scegliere o creare un AWS KMS key.
   - Seleziona Abilita protezione dall'eliminazione per impedire che un'operazione di eliminazione rimuova il cluster. Per impostazione predefinita, è selezionata la protezione dall'eliminazione.
4. (Facoltativo) In Tag, scegli o inserisci un tag per questo cluster.
5. Scegli Create cluster (Crea cluster).

Fase 2: Connect al cluster Aurora DSQL

Un endpoint del cluster viene generato automaticamente quando si crea un cluster Aurora DSQL in base all'ID e alla regione del cluster. Il formato di denominazione è. *clusterid*`.dsql.`*region*`.on.aws` Un client utilizza l'endpoint per creare una connessione di rete al cluster.

L'autenticazione è gestita tramite IAM, quindi non è necessario archiviare le credenziali nel database. Un token di autenticazione è una stringa unica di caratteri generata dinamicamente. Il token viene utilizzato solo per l'autenticazione e non influisce sulla connessione dopo che è stata stabilita. Prima di tentare la connessione, assicurati che la tua identità IAM disponga dell'`dsql:DbConnectAdmin`autorizzazione, come descritto inPrerequisiti.

> **Note**
>
> Per ottimizzare la velocità di connessione al database, usa il client PostgreSQL versione 17 e impostalo su direct:. `PGSSLNEGOTIATION` `PGSSLNEGOTIATION=direct`

Per connetterti al tuo cluster con un token di autenticazione

1. Nella console Aurora DSQL, scegli il cluster a cui desideri connetterti.
2. Scegli Connetti.

3. Copia l'endpoint da Endpoint (Host).
4. Assicurati che l'opzione Connect as admin sia selezionata nella sezione Token di autenticazione (Password).
5. Copia il token di autenticazione generato. Questo token è valido per 15 minuti.
6. Nella riga di comando del sistema operativo, utilizzare il comando seguente per avviare `psql` e connettersi al cluster. *`your_cluster_endpoint`*Sostituiscilo con l'endpoint del cluster che hai copiato in precedenza.

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

   Quando viene richiesta una password, inserisci il token di autenticazione che hai copiato in precedenza. Se tenti di riconnetterti utilizzando un token scaduto, la richiesta di connessione viene rifiutata. Per ulteriori informazioni, consulta Generazione di un token di autenticazione in Amazon Aurora DSQL.
7. Premere Invio. Dovresti vedere 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 negazione dell'accesso, consulta Risoluzione dei problemi di IAM e Come posso risolvere gli errori di accesso negato o non autorizzato con una policy IAM? .

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

Metti alla prova il tuo cluster Aurora DSQL eseguendo istruzioni SQL. Le seguenti istruzioni di esempio richiedono i file di dati denominati `department-insert-multirow.sql` and`invoice.csv`, che è possibile scaricare dal repository aurora-dsql-samplesaws-samples/. GitHub

Per eseguire comandi SQL di esempio in Aurora DSQL

1. Crea uno schema denominato. `example`

```
CREATE SCHEMA example;
```

2. Crea una tabella di fatture che utilizzi un UUID generato automaticamente come chiave primaria.

```
CREATE TABLE example.invoice(
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    created timestamp,
    purchaser int,
    amount float);
```

3. Crea un indice secondario che utilizzi la tabella vuota.

```
CREATE INDEX ASYNC invoice_created_idx on example.invoice(created);
```

4. Crea una tabella di reparto.

```
CREATE TABLE example.department(id INT PRIMARY KEY UNIQUE, name text, email text);
```

5. Usa il comando `psql \include` per caricare il file denominato `department-insert-multirow.sql` che hai scaricato dal repository aurora-dsql-samplesaws-samples/ su. GitHub Sostituisci *`my-path`* con il percorso della tua copia locale.

```
\include my-path/department-insert-multirow.sql
```

6. Usa il comando `psql \copy` per caricare il file denominato `invoice.csv` che hai scaricato dal repository aurora-dsql-samplesaws-samples/. GitHub Sostituisci *`my-path`* con il percorso della tua copia locale.

```
\copy example.invoice(created, purchaser, amount) from my-path/invoice.csv csv
```

7. Interroga i reparti e ordinali in base alle vendite totali.

```
SELECT name, sum(amount) AS sum_amount
FROM example.department LEFT JOIN example.invoice ON
 department.id=invoice.purchaser
GROUP BY name
HAVING sum(amount) > 0
```

```
ORDER BY sum_amount DESC;
```

Il seguente esempio di output mostra che il Dipartimento Tre registra il maggior numero di vendite.

```
           name           |     sum_amount
--------------------------+--------------------
 Example Department Three |  54061.67752854594
 Example Department Seven |  53869.65965365204
 Example Department Eight |  52199.73742066634
 Example Department One   | 52034.078869900826
 Example Department Six   |  50886.15556256385
 Example Department Two   |  50589.98422247931
 Example Department Five  | 49549.852635496005
 Example Department Four  |  49266.15578027619
(8 rows)
```

Fase 4: Creare un cluster multiregionale

Quando si crea un cluster multiregionale, si specificano le seguenti regioni:

Regione remota

Questa è la regione in cui si crea un secondo cluster. Crei un secondo cluster in questa regione e lo trasferisci al cluster iniziale. Aurora DSQL replica tutte le scritture sul cluster iniziale nel cluster remoto. È possibile leggere e scrivere su qualsiasi cluster.

Regione dei testimoni

Questa regione riceve tutti i dati scritti nel cluster multiregionale. Tuttavia, le regioni di riferimento non ospitano gli endpoint dei client e non forniscono l'accesso ai dati degli utenti. Una finestra limitata del registro delle transazioni crittografato viene mantenuta nelle regioni di riferimento. Questo registro facilita il ripristino e supporta il quorum transazionale se una regione non è più disponibile.

L'esempio seguente mostra come 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 di scrittura tra regioni e letture coerenti da entrambi gli endpoint regionali.

Per creare un cluster multiregionale

1. Accedi a AWS Management Console e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. Nel pannello di navigazione scegliere Clusters (Cluster).
3. Scegli Crea cluster e poi Multiregione.
4. (Facoltativo) Nelle impostazioni del cluster, seleziona una delle seguenti opzioni per il cluster iniziale:
   - Seleziona Personalizza le impostazioni di crittografia (avanzate) per scegliere o creare un AWS KMS key.
   - Seleziona Abilita protezione dall'eliminazione per impedire che un'operazione di eliminazione rimuova il cluster. Per impostazione predefinita, è selezionata la protezione dall'eliminazione.
5. Nelle impostazioni multiregionali, scegli le seguenti opzioni per il cluster iniziale:
   - In Witness Region, scegli una regione. Attualmente, solo le regioni con sede negli Stati Uniti sono supportate per le regioni di riferimento in cluster multiregionali.
   - (Facoltativo) Nell'ARN del cluster di regione remota, inserire un ARN per un cluster esistente in un'altra regione. Se non esiste alcun cluster che funga da secondo cluster nel cluster multiregionale, completa la configurazione dopo aver creato il cluster iniziale.
6. (Facoltativo) Scegliete i tag per il cluster iniziale.
7. Scegli Crea cluster per creare il tuo cluster iniziale. Se non hai inserito un ARN nel passaggio precedente, la console mostra la configurazione del cluster in attesa di notifica.
8. Nella finestra Configurazione del cluster in attesa di notifica, scegli Configurazione completa del cluster multiregionale. Questa azione avvia la creazione di un secondo cluster in un'altra regione.
9. Scegli una delle seguenti opzioni per il tuo 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 multiregionale.
   - Crea cluster in un'altra regione: scegli questa opzione per creare un secondo cluster. In Regione remota, scegli la regione per questo secondo cluster.
10. Scegli Crea cluster in ***your-second-region***, *your-second-region* dov'è la posizione del secondo cluster. La console si apre nella tua seconda regione.
11. (Facoltativo) Scegli le impostazioni del cluster per il secondo cluster. Ad esempio, puoi scegliere un AWS KMS key.

12. Scegli Crea cluster per creare il tuo secondo cluster.
13. Scegli Peer in ***initial-cluster-region***, *initial-cluster-region* dov'è la regione che ospita il primo cluster che hai creato.
14. Quando richiesto, scegli Conferma. Questo passaggio completa la creazione del cluster multiregionale.

Per connetterti al secondo cluster

1. Apri la console Aurora DSQL e scegli la regione per il tuo secondo cluster.
2. Scegli Cluster.
3. Seleziona la riga per il secondo cluster del cluster multiregionale.
4. In Azioni, scegli Apri in CloudShell.
5. Scegli Connect as admin.
6. Scegli Avvia CloudShell.
7. Seleziona Esegui.
8. Crea uno schema di esempio seguendo la procedura riportata di seguitoPassaggio 3: Esecuzione di comandi SQL di esempio in Aurora DSQL.

   Transazioni di esempio

   Example

   ```
   CREATE SCHEMA example;
   CREATE TABLE example.invoice(id UUID PRIMARY KEY DEFAULT gen_random_uuid(), created
    timestamp, purchaser int, amount float);
   CREATE INDEX ASYNC invoice_created_idx on example.invoice(created);
   CREATE TABLE example.department(id INT PRIMARY KEY UNIQUE, name text, email text);
   ```

9. Uso `psql` `copy` e `include` comandi per caricare dati di esempio. Per ulteriori informazioni, consulta Passaggio 3: Esecuzione di comandi SQL di esempio in Aurora DSQL.

   ```
   \copy example.invoice(created, purchaser, amount) from samples/invoice.csv csv
   \include samples/department-insert-multirow.sql
   ```

Per interrogare i dati nel secondo cluster dalla regione che ospita il cluster iniziale

1. Nella console Aurora DSQL, scegli la regione per il cluster iniziale.
2. Scegli Cluster.
3. Seleziona la riga per il secondo cluster del cluster multiregionale.
4. In Azioni, scegli Apri in CloudShell.
5. Scegli Connect as admin.
6. Scegli Avvia CloudShell.
7. Seleziona Esegui.
8. Interroga i dati che hai inserito nel secondo cluster.

   Example

   ```
   SELECT name, sum(amount) AS sum_amount
   FROM example.department
   LEFT JOIN example.invoice ON department.id=invoice.purchaser
   GROUP BY name
   HAVING sum(amount) > 0
   ORDER BY sum_amount DESC;
   ```

Autenticazione e autorizzazione per Aurora DSQL

Aurora DSQL utilizza i ruoli e le policy IAM per l'autorizzazione dei cluster. Associate i ruoli IAM ai ruoli del database PostgreSQL per l'autorizzazione del database. Questo approccio combina i vantaggi di IAM con i privilegi PostgreSQL. Aurora DSQL utilizza queste funzionalità per fornire una politica 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 l'autorizzazione:

Autenticazione IAM

Per autenticare la tua identità IAM quando gestisci i cluster Aurora DSQL, devi usare IAM. Puoi fornire l'autenticazione utilizzando AWS Management Console, o l'SDK AWS CLI.AWS

Autorizzazione IAM

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

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

Per ulteriori informazioni, consulta Utilizzo delle azioni politiche di IAM per gestire i cluster.

Connessione al cluster tramite IAM

Per connetterti al tuo cluster, usa IAM per l'autenticazione e l'autorizzazione:

Autenticazione IAM

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

Autorizzazione IAM

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

- `dsql:DbConnectAdmin`Usalo se stai usando il `admin` ruolo. Aurora DSQL crea e gestisce questo ruolo per te. Il seguente esempio di azione politica IAM consente di connettersi `admin` a. *`my-cluster`*

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

- `dsql:DbConnect`Utilizzalo se utilizzi un ruolo di database personalizzato. Puoi creare e gestire questo ruolo utilizzando i comandi SQL nel tuo 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 viene autorizzato per un massimo di un'ora per la connessione.

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 il ruolo. 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 `admin` ruolo e ruoli personalizzati. Aurora DSQL crea automaticamente un `admin` ruolo predefinito per te nel tuo cluster Aurora DSQL. Non puoi modificare il ruolo. `admin` Quando ti connetti al tuo database come`admin`, puoi emettere SQL per creare nuovi ruoli a livello di database da associare ai tuoi ruoli IAM. Per consentire ai ruoli IAM di connettersi al tuo database, associa i ruoli del database personalizzati ai ruoli IAM.

Autenticazione

Usa il `admin` ruolo per connetterti al tuo cluster. Dopo aver connesso il database, utilizza il comando `AWS IAM GRANT` per associare un ruolo di 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 ulteriori informazioni, consulta Autorizzazione dei ruoli del database a connettersi al cluster.

Autorizzazione

Usa il `admin` ruolo per connetterti al tuo cluster. Esegui i comandi SQL per configurare ruoli di database personalizzati e concedere autorizzazioni. Per ulteriori informazioni, consulta i ruoli del database PostgreSQL e i privilegi PostgreSQL nella documentazione di PostgreSQL.

Utilizzo delle azioni politiche IAM con Aurora DSQL

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

Utilizzo delle azioni politiche IAM per connettersi ai cluster

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

```
"dsql:DbConnectAdmin"
```

Quando ti connetti al cluster con un ruolo di database personalizzato, associa innanzitutto il ruolo IAM al ruolo del database. L'identità IAM che usi per connetterti al tuo cluster deve essere autorizzata a eseguire le seguenti azioni politiche IAM.

```
"dsql:DbConnect"
```

Per ulteriori informazioni sui ruoli personalizzati del database, consultaUtilizzo dei ruoli del database e dell'autenticazione IAM.

Utilizzo delle azioni politiche di IAM per gestire i cluster

Quando gestisci i cluster Aurora DSQL, specifica le azioni politiche 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 di ruolo solo alle autorizzazioni `GetCluster` and, come illustrato nella `ListClusters` seguente politica 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 politiche 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

Puoi revocare le autorizzazioni per i tuoi ruoli IAM per accedere ai ruoli a livello di database:

Revoca dell'autorizzazione dell'amministratore per la connessione ai cluster

Per revocare l'autorizzazione alla connessione al cluster con il `admin` ruolo, 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 dall'identità IAM, Aurora DSQL rifiuta tutti i nuovi tentativi di connessione da quell'identità IAM. Qualsiasi connessione attiva che utilizza l'identità IAM potrebbe rimanere autorizzata per tutta la durata della connessione. Per ulteriori informazioni sulla durata delle connessioni, consulta Quote e limiti.

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

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

Puoi anche rimuovere l'associazione tra il ruolo del database e IAM utilizzando il comando `AWS IAM REVOKE` nel tuo database. Per ulteriori informazioni sulla revoca dell'accesso dai ruoli del database, consulta. Revoca dell'autorizzazione del database da 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 di database personalizzati, vedi Privilegi 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 si crea 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 tuo client, puoi utilizzare lo stesso token di autenticazione se non è scaduto, oppure puoi 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 inConnessione 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 CloudShell Da 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. È possibile generare il token dalla console.

Per generare un token di autenticazione

1. Accedi a AWS Management Console e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. Scegli l'ID del cluster per il quale desideri generare un token di autenticazione. Se non hai ancora creato un cluster, segui i passaggi indicati in Fase 1: Creare un cluster Aurora DSQL a regione singola oFase 4: Creare un cluster multiregionale.
3. Scegli Connect, quindi seleziona Get Token.
4. Scegli se vuoi connetterti come `admin` o con un ruolo di database personalizzato.
5. Copia il token di autenticazione generato e usalo perAccesso ad Aurora DSQL tramite client SQL.

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

AWS CloudShell Da utilizzare per generare un token di autenticazione in Aurora DSQL

Prima di poter generare un token di autenticazione utilizzando AWS CloudShell, assicurati di eseguire le seguenti operazioni:

- Crea un cluster Aurora DSQL.
- Aggiungi un'autorizzazione per eseguire l'operazione Amazon S3 `get-object` per recuperare oggetti dall' Account AWS esterno dell'organizzazione. Per ulteriori informazioni, consulta la Amazon S3 User Guide.

Per generare un token di autenticazione utilizzando AWS CloudShell

1. Accedi a AWS Management Console 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. Segui Installazione o aggiornamento alla versione più recente di AWS CLI per installare il AWS CLI.

```
sudo ./aws/install --update
```

4. Esegui il comando seguente per generare un token di autenticazione per il `admin` ruolo. Sostituiscilo *`us-east-1`* con la tua regione e *`your_cluster_endpoint`* con l'endpoint del tuo cluster.

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

   ```
   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 funzionamento non autorizzato con una policy IAM? .

5. Usa il seguente comando da usare `psql` per avviare una connessione al tuo cluster.

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

6. Dovresti vedere una richiesta di immissione della password. Copia il token che hai generato e assicurati di non includere spazi o caratteri aggiuntivi. Incollalo nel seguente prompt da`psql`.

   ```
   Password for user admin:
   ```

7. Premere Invio. Dovresti vedere 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 negazione dell'accesso, consulta Risoluzione dei problemi di IAM e Come posso risolvere gli errori di accesso negato o non autorizzato con una policy IAM? .

Per ulteriori informazioni sui ruoli di 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 lo è`ACTIVE`, puoi generare un token di autenticazione sulla CLI utilizzando il `aws dsql` comando. Utilizzate una delle seguenti tecniche:

- Se ti stai connettendo al `admin` ruolo, usa l'`generate-db-connect-admin-auth-token`opzione.
- Se ti connetti con un ruolo di database personalizzato, usa l'`generate-db-connect-auth-token`opzione.

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

- *`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`*— Il Regione AWS, ad esempio `us-east-2` o`us-east-1`.

Gli esempi seguenti impostano l'ora di scadenza del token in 3600 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 `ACTIVE` stato. Gli esempi SDK utilizzano i seguenti attributi per generare un token di autenticazione per il `admin` ruolo:

- *`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`*) — Il luogo Regione AWS in cui si trova il cluster, ad esempio `us-east-2` o `us-east-1`.

Python SDK

È possibile generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`generate_db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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

Puoi generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`GenerateDBConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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

Puoi generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`getDbConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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

Puoi generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`generateDbConnectAdminAuthToken`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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

Puoi generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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

Puoi generare il token nei seguenti modi:

- Se ti stai connettendo con il `admin` ruolo, usa`generate_db_connect_admin_auth_token`.
- Se ti stai connettendo con un ruolo di database personalizzato, usa`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 invece necessario utilizzare`DSQLAuthTokenGenerator`, che è una classe di utilità. Il seguente esempio di codice mostra come generare il token di autenticazione per.NET.

È possibile generare il token nei seguenti modi:

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

L'esempio seguente utilizza la classe di `DSQLAuthTokenGenerator` utilità per generare il token di autenticazione per un utente con il `admin` ruolo. *`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 costruire manualmente la richiesta firmata, come mostrato nel seguente esempio di codice.

Nel seguente esempio di codice, specifica l'utente `action` based on the PostgreSQL:

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

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. L'utilizzo dei ruoli IAM fornisce credenziali di sicurezza temporanee. Puoi assumere temporaneamente un ruolo IAM in diversi modi:

- Cambiando ruolo in AWS Management Console
- 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 ulteriori informazioni sui metodi di utilizzo dei ruoli, consulta IAM Identities nella guida per l'utente IAM.

Utenti IAM

Un utente IAM è un'identità interna all'utente 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 autenticazione IAM, puoi utilizzare il ruolo IAM ARNs o l'utente IAM ARNs negli esempi seguenti.

Autorizzazione dei ruoli del database a connettersi al cluster

Crea un ruolo IAM e concedi l'autorizzazione alla connessione con l'azione politica IAM:`dsql:DbConnect`.

La policy IAM deve inoltre concedere il permesso di accedere alle risorse del cluster. Usa un wildcard (*) o segui le istruzioni in Uso delle 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 per connettersi al cluster.

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

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

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

   ```
   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 `GRANT` comando 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 ulteriori informazioni, consulta PostgreSQL e `GRANT` PostgreSQL Privileges nella documentazione di PostgreSQL.

Revoca dell'autorizzazione del database da 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 ulteriori 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, puoi utilizzare gli strumenti familiari di PostgreSQL beneficiando al contempo delle prestazioni e della scalabilità richieste per applicazioni distribuite moderne, native del cloud.

Aspetti salienti della compatibilità con PostgreSQL

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

Protocollo Wire

Aurora DSQL utilizza il protocollo cablato PostgreSQL v3 standard. 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 ulteriori informazioni, consulta 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 ulteriori informazioni, consulta Controllo della concorrenza in Aurora DSQL.

Principali differenze architettoniche

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 quanto segue:

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, vedere. 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 azioni:

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

Per ulteriori informazioni, consulta 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. Nota che Aurora DSQL si differenzia da PostgreSQL senza una clausola. `ORDER BY` Nelle sezioni seguenti, scopri il supporto di Aurora DSQL per i tipi di dati PostgreSQL e i comandi SQL.

Argomenti

Tipi di dati supportati in Aurora DSQL

Aurora DSQL supporta un sottoinsieme dei tipi PostgreSQL più comuni.

Argomenti

Tipi di dati numerici

Aurora DSQL supporta i seguenti tipi di dati numerici PostgreSQL.

Nome	Alias	Intervallo e precisione	Dimensioni dell'archiviazione	Supporto per l'indice
`smallint`	`int2`	Da -32768 a +32767	2 byte	Sì
`integer`	`int`, `int4`	Da -2147483648 a +2147483647	4 byte	Sì

Nome	Alias	Intervallo e precisione	Dimensioni dell'archiviazione	Supporto per l'indice
`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] L'impostazione predefinita 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 `CREATE TABLE` o`ALTER TABLE ADD COLUMN`, Aurora DSQL applica le impostazioni predefinite. Aurora DSQL applica dei limiti durante l'esecuzione `INSERT` delle istruzioni. `UPDATE`

Tipi di dati dei caratteri

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

Nome	Alias	Descrizione	Limite Aurora DSQL	Dimensioni dell'archiviazione	Supporto per gli indici
`character [ (n) ]`	`char [ (n) ]`	Stringa di caratteri a lunghezza fissa	4096 byte [1]	Variabile fino a 4100 byte	Sì
`character varying [ (n) ]`	`varchar [ (n) ]`	Stringa di caratteri a lunghezza variabile	65535 byte [1]	Variabile fino a 65539 byte	Sì

Nome	Alias	Descrizione	Limite Aurora DSQL	Dimensioni dell'archiviazione	Supporto per gli indici
`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 esegue `CREATE TABLE` o`ALTER TABLE ADD COLUMN`, Aurora DSQL applica le impostazioni predefinite. Aurora DSQL applica dei limiti durante l'esecuzione `INSERT` delle istruzioni. `UPDATE`

Tipi di dati data e ora

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

Nome	Alias	Descrizione	Intervallo	Risoluzione	Dimensioni dell'archiviazione	Supporto per gli indici
`date`		Data di calendario (anno, mese, giorno)	4713 A.C. — 5874897 D.C.	1 giorno	4 byte	Sì

Nome	Alias	Descrizione	Intervallo	Risoluzione	Dimensioni dell'archiviazione	Supporto per gli indici
`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`	`timestz`	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 diversi.

Nome	Alias	Descrizione	Limite Aurora DSQL	Dimensioni dell'archiviazione	Supporto per gli indici
`boolean`	`bool`	Booleani logici (true/false)		1 byte	Sì
`bytea`		Dati binari («array di byte»)	1 MiB [1]	Variabile fino al limite di 1 MiB	No
`UUID`		Identificatore universalmente unico		16 byte	Sì

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

Tipi di dati di esecuzione 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 quelli `varchar` `integer` che definisci nel tuo 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 l'esecuzione della query:

Tipo di array

Aurora DSQL supporta matrici dei tipi di dati supportati. Ad esempio, è possibile avere una matrice di numeri interi. La funzione `string_to_array` divide una stringa in un array in stile PostgreSQL con il delimitatore di 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 una risposta simile alla 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 analizzare i log, filtrare le sottoreti IP o eseguire calcoli di rete all'interno di una query. Per ulteriori informazioni, vedere inet nella documentazione di PostgreSQL.

SQL supportato per Aurora DSQL

Aurora DSQL supporta un'ampia gamma di funzionalità SQL di base di PostgreSQL. Nelle sezioni seguenti, puoi scoprire il supporto generale delle espressioni PostgreSQL. L'elenco non è completo.

Comando **SELECT**

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

Clausola principale	Clausole supportate
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	
`WITH`(espressioni di tabella comuni)	

Clausola principale	Clausole supportate
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`
`OVER`	`RANK ()`, `PARTITION BY`
`FOR UPDATE`	

DDL (Data Definition Language)

Aurora DSQL supporta i seguenti comandi PostgreSQL DDL.

Comando	Clausola principale	Clausole supportate
`CREATE`	`TABLE`	Per informazioni sulla sintassi supportat a del `CREATE TABLE` comando, vedere. `CREATE TABLE`
`ALTER`	`TABLE`	Per informazioni sulla sintassi supportat a del `ALTER TABLE` comando, vedere. `ALTER TABLE`
`DROP`	`TABLE`	
`CREATE`	`[UNIQUE] INDEX ASYNC`	È possibile utilizzare questo comando con i seguenti parametri:`ON`,`NULLS FIRST`,`NULLS LAST`.

Comando	Clausola principale	Clausole supportate
		Per informazioni sulla sintassi supportat a del `CREATE INDEX ASYNC` comando, vedereIndici asincroni in Aurora SQL.
`DROP`	`INDEX`	
`CREATE`	`VIEW`	Per ulteriori informazioni sulla sintassi supportata del `CREATE VIEW` comando, vedere. `CREATE VIEW`
`ALTER`	`VIEW`	Per informazioni sulla sintassi supportata del `ALTER VIEW` comando, vedere. ALTER VIEW
`DROP`	`VIEW`	Per informazioni sulla sintassi supportata del `DROP VIEW` comando, vedere. `DROP VIEW`
`CREATE`	`ROLE`, `WITH`	
`CREATE`	`FUNCTION`	`LANGUAGE SQL`
`CREATE`	`DOMAIN`	

Linguaggio di manipolazione dei dati (DML)

Aurora DSQL supporta i seguenti comandi DML PostgreSQL.

Comando	Clausola principale	Clausole supportate
`INSERT`	`INTO`	`VALUES` `SELECT`

`UPDATE`	`SET`	`WHERE (SELECT)`
		
		`FROM, WITH`

Comando	Clausola principale	Clausole supportate
DELETE	FROM	USING, WHERE

Linguaggio di controllo dei dati (DCL)

Aurora DSQL supporta i seguenti comandi DCL PostgreSQL.

Comando	Clausole supportate
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Linguaggio di controllo delle transazioni (TCL)

Aurora DSQL supporta i seguenti comandi TCL PostgreSQL.

Comando	Clausole supportate
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

Comandi di utilità

Aurora DSQL supporta i seguenti comandi di utilità 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 TABLEmodifica 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 VIEWdefinisce una nuova vista persistente. Aurora DSQL non supporta le visualizzazioni 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 ]
```

Descrizione

`CREATE VIEW`definisce una visualizzazione di un'interrogazione. La vista non è materializzata fisicamente. La query viene invece eseguita ogni volta che viene fatto riferimento alla vista in un'interrogazione.

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

Se viene fornito un nome di schema, ad esempio`CREATE VIEW myschema.myview ...`), la vista viene creata nello schema specificato. Altrimenti, viene creata nello schema corrente.

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

Parametri

`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;`

È necessario specificare un elenco di nomi di colonne di visualizzazione per una vista ricorsiva.

`name`

Il nome della vista da creare, che può essere facoltativamente qualificata dallo schema. È necessario specificare un elenco di nomi di colonna per una vista ricorsiva.

`column_name`

Un elenco opzionale di nomi da utilizzare per le colonne della vista. Se non viene fornito, i nomi delle colonne vengono dedotti 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 essere uno dei due `local` ed è equivalente a specificare. `cascaded` `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 delle condizioni della vista (e di tutte `WHERE` le condizioni che utilizzano operatori contrassegnati come`LEAKPROOF`) a essere valutate per prime.
- `security_invoker (boolean)`—Questa opzione fa sì che le relazioni di base sottostanti vengano confrontate con i privilegi dell'utente della vista anziché del proprietario della vista. Per tutti i dettagli, consulta le note riportate di seguito.

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

`query`

Un `VALUES` comando `SELECT` or che fornirà 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, `INSERT` i `UPDATE` comandi 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 non `CHECK OPTION` è specificato, `INSERT` i `UPDATE` comandi della vista possono creare righe che non sono visibili attraverso la vista. Sono supportate le seguenti opzioni di controllo.
- `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 non specifichino anche il`CHECK OPTION`).
- `CASCADED`: le nuove righe vengono verificate rispetto alle condizioni della vista e di tutte le viste di base sottostanti. Se `CHECK OPTION` viene specificato e non viene specificato né l'uno `LOCAL` né `CASCADED` l'altro, `CASCADED` viene assunto.

> **Note**
>
> Non `CHECK OPTION` può essere utilizzato con le `RECURSIVE` viste. `CHECK OPTION`È supportato solo nelle visualizzazioni che sono aggiornabili automaticamente.

Note

Utilizza l'`DROP VIEW`istruzione per eliminare le visualizzazioni.

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 è predefinito su. `?column?;` Inoltre, il tipo di dati della colonna è predefinito`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 visualizzazioni sono protette dalla manomissione.

- Se la `security_invoker` proprietà della vista è impostata su true, l'accesso alle relazioni di base sottostanti è determinato dalle autorizzazioni dell'utente che esegue la query, anziché dal proprietario della vista. Pertanto, l'utente di una vista 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 Security Invoker, verrà trattata come se fosse stato effettuato l'accesso direttamente dalla query originale. Pertanto, una vista 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 richiamate nella vista vengono trattate come se fossero state chiamate direttamente dalla query utilizzando la vista. Pertanto, l'utente di una vista deve disporre delle autorizzazioni per richiamare 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 invoca, non il proprietario della

vista. Ciò non è influenzato dall'`security_invoker`impostazione della vista, quindi una vista `security_invoker` impostata su false non è equivalente a una `SECURITY DEFINER` funzione.

- L'utente che crea o sostituisce una vista deve disporre `USAGE` dei privilegi su tutti gli schemi a cui si fa riferimento nella query di visualizzazione, per poter cercare gli oggetti di 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 solo il `USAGE` privilegio sullo schema che contiene la vista, non sugli schemi a cui si fa riferimento nella query di visualizzazione, anche per una vista Security Invoker.
- Quando `CREATE OR REPLACE VIEW` viene utilizzata in una vista esistente, vengono modificate solo la `SELECT` regola di definizione della vista, più eventuali `WITH ( ... )` parametri e relativi parametri. `CHECK OPTION` Le altre proprietà della vista, tra cui proprietà, autorizzazioni e regole non selezionate, rimangono invariate. Devi essere il proprietario della vista per sostituirla (questo include essere un membro del ruolo proprietario).

Visualizzazioni aggiornabili

Le viste semplici sono aggiornabili automaticamente: il sistema consentirà `INSERT` di utilizzare le `DELETE` istruzioni nella vista nello stesso modo in cui si utilizza una tabella normale. `UPDATE` Una vista è aggiornabile automaticamente se soddisfa tutte le seguenti condizioni:

- La vista deve avere esattamente una voce nell'`FROM`elenco, che deve essere una tabella o un'altra vista aggiornabile.
- La definizione della vista non deve contenere `WITH`,`DISTINCT`,`GROUP BY`, `HAVING`,`LIMIT`, o `OFFSET` clausole di primo livello.
- La definizione della vista non deve contenere operazioni di set (`UNION`,`INTERSECT`, o`EXCEPT`) al livello superiore.
- L'elenco di selezione della vista non deve contenere aggregati, funzioni di finestra o funzioni di restituzione dei set.

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'`UPDATE`istruzione `INSERT` or tenta di assegnarle un valore.

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

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

Puoi usare il `CHECK OPTION` per impedire che `INSERT` i `UPDATE` comandi creino righe che non sono visibili attraverso la vista.

Se una vista aggiornabile automaticamente è contrassegnata con la proprietà security_barrier, tutte le `WHERE` condizioni 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. Nota che per questo motivo, le righe che alla fine non vengono restituite (perché non soddisfano `WHERE` le condizioni dell'utente) potrebbero comunque finire per essere bloccate. Puoi usarlo `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 l'inserimento, l'aggiornamento o l'eliminazione nella 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 security_invoker impostato su true, l'utente che esegue l'aggiornamento, anziché il proprietario della vista, 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';
```

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

Crea una vista con`LOCAL CHECK OPTION`.

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

Questo creerà una vista che controlla sia la `kind` riga che quella `classification` delle nuove righe.

Crea una vista con un mix 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 visualizzazione supporterà`INSERT`, e`UPDATE`. `DELETE` Tutte le colonne della tabella dei film saranno aggiornabili, mentre le colonne calcolate `avg_rating` saranno di `country` 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 in base allo schema, il suo autoreferenziamento interno non è qualificato dallo schema. `CREATE` Questo perché il nome di 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 delle barriere di sicurezza e le viste degli invoker di sicurezza. Aurora DSQL supporta queste estensioni linguistiche.

ALTER VIEW

L'`ALTER VIEW`istruzione 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 [, ... ] )
```

Descrizione

`ALTER VIEW`modifica varie proprietà ausiliarie di una vista. (Se vuoi modificare la query di definizione della vista, usa`CREATE OR REPLACE VIEW`.) È necessario possedere la vista da utilizzare`ALTER VIEW`. Per modificare lo schema di una vista, devi anche disporre dei `CREATE` privilegi sul nuovo schema. Per modificare il proprietario, devi essere in grado di `SET ROLE` ricoprire il nuovo ruolo di proprietario e quel ruolo deve avere `CREATE` i privilegi sullo schema della vista. Queste restrizioni

impongono che la modifica del proprietario non comporti alcun effetto che non si possa fare eliminando e ricreando la visualizzazione.)

Parametri

Parametri `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 `UPDATE` comando `INSERT` o 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 visualizzazione.

new_schema

Il nuovo schema per la vista.

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

Imposta o reimposta un'opzione di visualizzazione. Le opzioni supportate sono le seguenti.

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

- `security_invoker (boolean)`- Modifica la proprietà della barriera di sicurezza della vista. Il valore deve essere un valore booleano, ad esempio o. `true` `false`

Note

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

Esempi

Rinominare la vista in. `foo` `bar`

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

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

Sintassi supportata

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

Descrizione

`DROP VIEW`elimina una vista esistente. Per eseguire questo comando devi essere il proprietario della vista.

Parametri

`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`

Elimina 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'`IF EXISTS`opzione, che è un'estensione PostgreSQL supportata da Aurora DSQL.

Funzionalità PostgreSQL non supportate in Aurora SQL

Aurora DSQL è compatibile con PostgreSQL. Ciò significa che Aurora DSQL supporta funzionalità relazionali di base come transazioni ACID, indici secondari, join, inserti e aggiornamenti. Per una panoramica delle funzionalità SQL supportate, consulta Espressioni SQL supportate.

Le sezioni seguenti evidenziano quali funzionalità di PostgreSQL non sono attualmente supportate in Aurora DSQL.

Oggetti non supportati

Gli oggetti non supportati da Aurora DSQL includono quanto segue:

- Più database su un singolo cluster Aurora DSQL
- Tabelle temporanee
- Trigger
- Tipi (supporto parziale)
- Spazi tabelle
- Funzioni scritte in linguaggi diversi da SQL
- Sequenze
- Partizioni

Vincoli non supportati

- Chiavi esterne
- Vincoli di esclusione

Comandi non supportati

- `ALTER SYSTEM`
- `TRUNCATE`
- `SAVEPOINT`
- `VACUUM`

> **Note**
>
> Aurora DSQL non richiede l'aspirazione. Il sistema mantiene le statistiche e gestisce automaticamente l'ottimizzazione dello storage senza comandi manuali di vacuum.

Estensioni non supportate

Aurora DSQL non supporta le estensioni PostgreSQL. La tabella seguente mostra le estensioni che non sono supportate:

- `PL/pgSQL`
- `PostGIS`

- PGVector
- PGAudit
- Postgres_FDW
- PGCron
- pg_stat_statements

Espressioni SQL non supportate

La tabella seguente descrive le clausole che non sono supportate in Aurora DSQL.

Categoria	Clausola principale	Clausola non supportata
CREATE	INDEX ASYNC	ASC | DESC
CREATE	INDEX[1]	
TRUNCATE		
ALTER	SYSTEM	Tutti ALTER SYSTEM i comandi sono bloccati.
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION
CREATE	FUNCTION	LANGUAGE *non-sql-lang* , *non-sql-lang* dov'è una lingua diversa da SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	
CREATE	TRIGGER	

Categoria	Clausola principale	Clausola non supportata
CREATE	TYPE	
CREATE	DATABASE	Non è possibile creare database aggiuntivi.

[1] Vedere Indici asincroni in Aurora SQL per creare un indice su una colonna di una tabella specificata.

Considerazioni su Aurora DSQL per la compatibilità con PostgreSQL

Considerate le seguenti limitazioni di compatibilità quando utilizzate Aurora DSQL. Per considerazioni generali, vedere. Considerazioni sull'utilizzo di Amazon Aurora DSQL Per quote e limiti, vedere. Quote di cluster e limiti del database in Amazon Aurora SQL

- Aurora DSQL utilizza un unico database integrato denominato. `postgres` Non è possibile creare database aggiuntivi o rinominare o eliminare il database. `postgres`
- Il `postgres` database utilizza la codifica dei caratteri UTF-8. Non è possibile modificare la codifica.
- Il database utilizza solo le regole di `C` confronto.
- Aurora DSQL utilizza `UTC` come fuso orario del sistema. Non è possibile modificare il fuso orario utilizzando parametri o istruzioni SQL come. `SET TIMEZONE`
- Il livello di isolamento delle transazioni è fisso in PostgreSQL`Repeatable Read`.
- Le transazioni hanno i 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. `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 `GRANT [permission] on DATABASE` comando e concedere

`CREATE` autorizzazioni 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 `CREATE` autorizzazioni allo schema pubblico stesso. Gli utenti non amministratori devono utilizzare schemi diversi creati dall'utente per la creazione di oggetti.
- Aurora DSQL non supporta il comando. `ALTER ROLE [] CONNECTION LIMIT` Contatta l' AWS assistenza se hai bisogno di aumentare il limite di connessione.
- Aurora DSQL non supporta asyncpg, il driver di database PostgreSQL asincrono per Python.

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 delle istantanee, 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 rendimento 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.

Progetta le tue applicazioni per implementare la logica dei tentativi per gestire i conflitti. Il modello di progettazione ideale è idempotente e consente di ripetere la transazione come prima risorsa, quando possibile. La logica consigliata è simile alla logica di interruzione e riprova 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, riduci al minimo la contesa elevata su chiavi singole o intervalli di chiavi piccoli. 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 tue tabelle.
- Evita gli schemi che aumentano 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 Data Definition Language (DDL) si comporta in modo diverso in Aurora SQL rispetto a PostgreSQL. Aurora DSQL offre un livello di database Multi-AZ distribuito e senza condivisione basato su flotte di elaborazione e storage 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 allo schema DDL 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. Ad esempio, si consideri 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 entrambe le `CREATE` istruzioni. `INSERT`

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

DDL asincrono

In PostgreSQL standard, le operazioni DDL `CREATE INDEX` come 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, DDL su tabelle di grandi dimensioni può essere eseguito senza tempi di inattività o impatto sulle prestazioni. Per ulteriori informazioni sul job manager asincrono in Aurora DSQL, vedere. 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'`CLUSTER`operazione 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 principale 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'`CLUSTER`operazione 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 aiuta Aurora DSQL a scalare automaticamente e a gestire i carichi di lavoro simultanei in modo efficiente.

Linee guida per la scelta di una chiave primaria

Quando scegli e utilizzi una chiave primaria in Aurora DSQL, considera le seguenti linee guida:

- Definite una chiave primaria quando create una tabella. Non puoi modificare questa chiave o 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 ridimensionamento automatico della velocità di scrittura. Se non si specifica una chiave primaria, Aurora DSQL assegna un ID nascosto sintetico.
- Per le tabelle con volumi di scrittura elevati, evita di utilizzare numeri interi che aumentano in modo monotono come chiavi primarie. Ciò può causare problemi di prestazioni se si indirizzano tutti i nuovi inserti su un'unica partizione. Utilizzate 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, puoi utilizzare una chiave crescente. Esempi di tasti ascendenti 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 prestazioni, scegli 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 ulteriori 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 ulteriori informazioni, consulta Limiti del database in Aurora DSQL e Tipi di dati supportati in Aurora DSQL.

Indici asincroni in Aurora SQL

Il `CREATE INDEX ASYNC` comando 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`CREATE INDEX ASYNC`, Aurora DSQL restituisce immediatamente un. `job_id`

È possibile monitorare lo stato di questo lavoro asincrono utilizzando la visualizzazione 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'`*

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

`DROP INDEX`

Annulla un processo di creazione dell'indice in corso.

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 questo momento, potresti 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 ]
```

Parametri

`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 specificate 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`

Il 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`

Speciifica 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 `CREATE INDEX ASYNC` comando 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, la `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* procedura è utile. Assicura 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 `sys.jobs` visualizzazione ed elimina le attività con uno stato pari o superiore a 30 minuti. `completed` `failed` Pertanto, mostra `sys.jobs` 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. `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. Create una 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. Inserite una riga 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 `CREATE INDEX` comando restituisce un ID di lavoro, come illustrato di seguito.

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

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

Interrogazione dello stato di creazione dell'indice: esempio

Eseguite una query nella vista di `sys.jobs` sistema 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 dello stato può corrispondere a 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'operazione non è riuscita. Per ulteriori informazioni, consulta la colonna dei dettagli. Se Aurora DSQL non è riuscita a creare l'indice, Aurora DSQL non rimuove automaticamente la definizione dell'indice. È necessario rimuovere manualmente l'indice con il comando. `DROP INDEX`	Aurora DSQL

È inoltre possibile interrogare lo stato dell'indice tramite le tabelle `pg_index` del catalogo e. `pg_class` In particolare, `indisvalid` gli attributi `indisimmediate` possono dirti in che stato si trova il tuo indice. Sebbene Aurora DSQL crei l'indice, lo stato iniziale è di. `INVALID` Il `indisvalid` flag dell'indice restituisce `FALSE` `of`, che indica che l'indice non è valido. Se il flag restituisce `TRUE` `ot`, 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 compilazione dell'indice univoco

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

Per 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. Eliminate l'indice fallito.
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'unicità 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 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
);
```

Inserisci 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');
```

Esegui una query di rilevamento duplicata

```
-- 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;
```

Visualizza 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, 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 PostgreSQL

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

Nome	Descrizione
`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 visualizzazione delle 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.

Nome	Applicabile a Aurora DSQL
`pg_aggregate`	No
`pg_am`	Sì
`pg_amop`	No

Nome	Applicabile a Aurora DSQL
pg_amproc	No
pg_attrdef	Sì
pg_attribute	Sì
pg_authid	No (uso) 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
pg_foreign_data_wrapper	No
pg_foreign_server	No

Nome	Applicabile a Aurora DSQL
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
pg_range	Sì
pg_replication_origin	No

Nome	Applicabile a Aurora DSQL
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
pg_ts_template	Sì
pg_type	Sì

Nome	Applicabile a Aurora DSQL
pg_user_mapping	No

Visualizzazioni di sistema supportate e non supportate

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

Nome	Applicabile a 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
pg_prepared_statements	No
pg_prepared_xacts	No

Nome	Applicabile a Aurora DSQL
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
pg_views	Sì
pg_stat_activity	No

Nome	Applicabile a Aurora DSQL
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
pg_statio_all_sequences	No
pg_stat_slru	No

Nome	Applicabile a Aurora DSQL
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
pg_statio_user_indexes	No

Le viste sys.jobs e sys.iam_pg_role_mappings

Aurora DSQL supporta le seguenti visualizzazioni di sistema:

`sys.jobs`

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

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

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

La tabella pg_class

La `pg_class` tabella memorizza i metadati sugli oggetti del database. Per ottenere il conteggio approssimativo di quante righe ci sono in una tabella, esegui il comando seguente.

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

Il comando restituisce un output simile al seguente:

```
  reltuples
--------------
```

```
9.993836e+08
```

Il comando **ANALYZE**

Il ANALYZE comando raccoglie statistiche sul contenuto delle tabelle nel database e archivia i risultati nella vista di pg_stats sistema. Successivamente, 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 ANALYZE comando all'interno di una transazione esplicita. ANALYZEnon 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 viene eseguito 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.

ANALYZEviene eseguito in modo asincrono in background e la sua 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**
>
> ANALYZEi lavori vengono fatturati come gli altri lavori asincroni in Aurora DSQL. Quando si modifica una tabella, ciò può attivare indirettamente un processo automatico di raccolta di statistiche in background, che può comportare costi di misurazione dovuti all'attività associata a livello di sistema.

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

Gestione dei cluster Aurora DSQL

Aurora DSQL offre diverse opzioni di configurazione per aiutarti a stabilire l'infrastruttura di database giusta per le tue esigenze. 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

Creazione di un cluster

Crea un cluster utilizzando il create-cluster comando.

> **Note**
>
> La creazione di cluster è un'operazione asincrona. Chiama l'GetClusterAPI fino a quando lo stato non cambia a. `ACTIVE` Puoi connetterti al tuo cluster dopo che è diventato attivo.

Example Comando

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

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

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

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

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

Aggiorna un cluster esistente utilizzando il update-cluster comando.

> **Note**
>
> Gli aggiornamenti sono operazioni asincrone. Chiama l'GetClusterAPI fino a quando lo stato non cambia per visualizzare `ACTIVE` le modifiche.

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

Elimina un cluster esistente utilizzando il delete-cluster comando.

> **Note**
>
> È possibile eliminare solo i cluster con la protezione dall'eliminazione disattivata. Per impostazione predefinita, la protezione dall'eliminazione è abilitata quando si creano nuovi cluster.

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

Elenca i tuoi cluster utilizzando il list-clusters comando.

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 multiregionali

Questo capitolo spiega come configurare e gestire i cluster su più cluster. Regioni AWS

Connessione al cluster multiregionale

I cluster peer multiregionali forniscono due endpoint regionali, uno per ogni cluster peer. Regione AWS 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 peer, un cluster multiregionale 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 in più regioni. Le regioni di riferimento multiregionali non dispongono di endpoint.

Creazione di cluster multiregionali

Per creare cluster multiregionali, è innanzitutto necessario creare un cluster con una regione di riferimento. Quindi esegui il peering di questo cluster con un secondo cluster che condivide la stessa regione di riferimento del primo cluster. L'esempio seguente mostra come creare cluster negli Stati Uniti orientali (Virginia settentrionale) e negli Stati Uniti orientali (Ohio) con gli Stati Uniti occidentali (Oregon) come regione di riferimento.

Fase 1: creare un cluster negli Stati Uniti orientali (Virginia settentrionale)

Per creare un cluster negli Stati Uniti orientali (Virginia settentrionale) Regione AWS con proprietà multiregionali, usa il comando seguente.

```
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 attiva `PENDING_SETUP` finché non si aggiorna il cluster con l'ARN del relativo cluster peer.

Fase 2: Creare il secondo cluster negli Stati Uniti orientali (Ohio)

Per creare un cluster negli Stati Uniti orientali (Ohio) Regione AWS con proprietà multiregionali, usa 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 `PENDING_SETUP` stato fino a quando non viene aggiornata con l'ARN di un altro cluster per il peering.

Fase 3: cluster peer negli Stati Uniti orientali (Virginia settentrionale) con Stati Uniti orientali (Ohio)

Per effettuare il peering tra il cluster degli Stati Uniti orientali (Virginia settentrionale) e il cluster degli Stati Uniti orientali (Ohio), usa il comando. `update-cluster` Specificate il nome del cluster US East (Virginia settentrionale) e una stringa JSON con l'ARN del cluster US East (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: cluster peer negli Stati Uniti orientali (Ohio) con Stati Uniti orientali (Virginia settentrionale)

Per effettuare il peering tra il cluster degli Stati Uniti orientali (Ohio) e il cluster degli Stati Uniti orientali (Virginia settentrionale), usa il comando. `update-cluster` Specificate il nome del cluster US East (Ohio) e una stringa JSON con l'ARN del cluster US East (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**
>
> Dopo il peering riuscito, entrambi i cluster passano dallo stato «PENDING_SETUP» a «CREATING» e infine allo stato «ACTIVE» quando sono pronti per l'uso.

Visualizzazione delle proprietà dei cluster multiregionali

Quando si descrive un cluster, è possibile visualizzare le proprietà multiregionali per i cluster in diverse aree. 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"
       ]
```

```
    }
}
```

Cluster peer 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"]}'
```

Ciò 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 PENDING_SETUP a CREATING e infine a ACTIVE quando sono pronti per l'uso.

Eliminazione di cluster multiregionali

Per eliminare un cluster multiregionale, è necessario completare due passaggi.

1. Disattiva la protezione dall'eliminazione per ogni cluster.
2. Elimina ogni cluster peered separatamente nei rispettivi Regione AWS

Aggiorna ed elimina il cluster negli Stati Uniti orientali (Virginia settentrionale)

1. Disattiva la protezione da eliminazione utilizzando il `update-cluster` comando.

   ```
   aws dsql update-cluster \
     --region us-east-1 \
     --identifier 'foo0bar1baz2quux3quuxquux4' \
     --no-deletion-protection-enabled
   ```

2. Eliminare il cluster utilizzando il `delete-cluster` comando.

```
aws dsql delete-cluster \
  --region us-east-1 \
  --identifier 'foo0bar1baz2quux3quuxquux4'
```

Questo comando restituisce la risposta seguente.

```
{
    "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 `PENDING_DELETE` stato. L'eliminazione non è completa finché non elimini il cluster peered negli Stati Uniti orientali (Ohio).

Aggiorna ed elimina il cluster negli Stati Uniti orientali (Ohio)

1. Disattiva la protezione da eliminazione utilizzando il `update-cluster` comando.

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quux4quuux' \
--no-deletion-protection-enabled
```

2. Eliminare il cluster utilizzando il `delete-cluster` comando.

```
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 `PENDING_DELETE` stato. Dopo alcuni secondi, il sistema passa automaticamente allo stato di entrambi i cluster peered dopo la convalida. `DELETING`

Configurazione di cluster multiregionali utilizzando AWS CloudFormation

È possibile utilizzare la stessa AWS CloudFormation risorsa `AWS::DSQL::Cluster` per distribuire e gestire cluster Aurora DSQL a regione singola e multiarea.

Consulta il riferimento sul tipo di risorsa DSQL di Amazon Aurora per ulteriori informazioni su come creare, modificare e gestire i cluster utilizzando la risorsa. `AWS::DSQL::Cluster`

Creazione della configurazione iniziale del cluster

Innanzitutto, crea un AWS CloudFormation modello per definire il tuo cluster multiregionale:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
```

Crea pile in entrambe le regioni utilizzando i seguenti comandi AWS CLI:

```
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
```

Ricerca di identificatori di cluster

Recupera la risorsa fisica IDs per i tuoi 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 del cluster

Aggiorna il AWS CloudFormation modello per includere entrambi i cluster ARNs:

```
---
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
```

Applica 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
```

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)

È possibile creare e gestire le risorse utilizzando la shell della riga di comando. AWS CLI AWS CLI Fornisce l'accesso diretto al modulo APIs Servizi AWS, come Aurora DSQL. Per la sintassi e gli esempi dei comandi per Aurora DSQL, vedere dsql nella Guida ai comandi.AWS CLI

AWS kit di sviluppo software () SDKs

AWS fornisce 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 SQL Aurora

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 Riferimento API.

AWS CloudFormation

AWS::DSQL::ClusterSi tratta di una AWS CloudFormation risorsa che consente di creare e gestire i cluster Aurora DSQL come parte dell'infrastruttura sotto forma di codice. AWS CloudFormation ti 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 AWS CloudFormation modelli, puoi effettuare il provisioning dichiarativo dei cluster Aurora DSQL insieme alle altre risorse cloud. Questo aiuta a garantire che l'infrastruttura di dati venga distribuita e gestita insieme al resto dello stack di applicazioni.

Amazon Aurora DSQL SDKs, driver e codice di esempio

AWS i kit di sviluppo software (SDKs) sono disponibili per molti dei linguaggi di programmazione più diffusi. Ogni SDK fornisce un'API, esempi di codice, e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Adattatori e dialetti

La tabella seguente mostra gli adattatori ORM e i dialetti di database disponibili per Aurora DSQL.

Linguaggio di programmazione	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/

Esempi di codice

Gestione dei cluster tramite SDK AWS

La tabella seguente mostra esempi di codice di gestione dei cluster utilizzati per diversi linguaggi di programmazione. AWS SDKs

Linguaggio di programmazione	Link al repository di esempio
C++	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/cpp/cluster
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/dotnet/cluster
Go	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/go/cluster

Linguaggio di programmazione	Link al repository di esempio
Java	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/java/cluster
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/javascript/cluster
Python	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/python/cluster
Ruby	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/ruby/cluster
Rust	https://github.com/aws-samples/aurora-dsql-samples/_gestione tree/main/rust/cluster

Driver ed esempi di Object-Relational Mapping () ORMs

La tabella seguente mostra esempi di driver di database e framework ORM per diversi linguaggi di programmazione.

Linguaggio di programmazione	Driver o framework	Link al repository di esempio
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

Linguaggio di programmazione	Driver o framework	Link al repository di esempio
Java	Ibernazione	https://github.com/awslabs/aurora-dsql-hibernate/-app clinica tree/main/examples/pet
Java	pgJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	nodo postgres	https://github.com/aws-samples/aurora-dsql-samples/-postgres 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
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/tree/main/examples/pet-app clinica
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg

Linguaggio di programmazione	Driver o framework	Link al repository di esempio
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx
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/-ormtree/main/typescript/type

Aurora DSQL con AWS CLI

Consulta le sezioni seguenti per scoprire come gestire i cluster con. AWS CLI

CreateCluster

Per creare un cluster, usa il `create-cluster` comando.

> **Note**
>
> La creazione del cluster avviene in modo asincrono. Chiama l'`GetCluster`API fino a quando lo stato non è raggiunto. `ACTIVE` Puoi connetterti a un cluster una volta che lo diventa`ACTIVE`.

Comando di esempio

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

> **Note**
>
> Se desideri disabilitare la protezione dall'eliminazione al momento della creazione, includi il `--no-deletion-protection-enabled` flag.

Risposta di esempio

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "CREATING",
    "creationTime": "2025-05-22T14:03:26.631000-07:00",
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    },
    "deletionProtectionEnabled": true
}
```

GetCluster

Per descrivere un cluster, usa il `get-cluster` comando.

Comando di esempio

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

Risposta di esempio

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2025-05-22T14:03:26.631000-07:00",
    "deletionProtectionEnabled": true,
    "tags": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

UpdateCluster

Per aggiornare un cluster esistente, utilizzare il `update-cluster` comando.

> **Note**
>
> Gli aggiornamenti avvengono in modo asincrono. Chiama l'`GetCluster`API fino allo stato stabilito `ACTIVE` e osserverai le modifiche.

Comando di esempio

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

Risposta di esempio

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

DeleteCluster

Per eliminare un cluster esistente, utilizzare il `delete-cluster` comando.

> **Note**
>
> È possibile eliminare solo un cluster con la protezione dall'eliminazione disattivata. La protezione da eliminazione è abilitata per impostazione predefinita durante la creazione di nuovi cluster.

Comando di esempio

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

Risposta di esempio

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

ListClusters

Per ottenere l'a dei cluster, usa il `list-clusters` comando.

Comando di esempio

```
aws dsql list-clusters --region us-east-1
```

Risposta di esempio

```
{
    "clusters": [
        {
            "identifier": "abc0def1baz2quux3quux4quuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuux"
        },
        {
            "identifier": "abc0def1baz2quux3quux4quuuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuuux"
        }
    ]
}
```

GetCluster su cluster multiregionali

Per ottenere informazioni su un cluster multiregionale, utilizzare il comando. `get-cluster` Per i cluster multiregionali, la risposta includerà il cluster collegato. ARNs

Comando di esempio

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

```
  --identifier your_cluster_id
```

Risposta di esempio

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2025-05-22T13:56:18.716000-07:00",
    "deletionProtectionEnabled": true,
    "multiRegionProperties": {
        "witnessRegion": "us-west-2",
        "clusters": [
            "arn:aws:dsql:us-east-1:842685632318:cluster/fuabuc7d3szkr37uqd5znkjynu"
        ]
    },
    "tags": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

Crea, leggi, aggiorna ed elimina i cluster Aurora DSQL

Vengono forniti esempi di creazione, lettura, aggiornamento, eliminazione (CRUD) sia per le distribuzioni a regione singola che a più regioni. È inclusa una `cluster_management` sezione dedicata per ogni linguaggio di programmazione che illustra queste attività di gestione chiave.

Le implementazioni in un'unica regione sono ideali per le applicazioni che servono gli utenti in un'area geografica specifica, offrendo una gestione semplificata e una latenza inferiore. Le implementazioni multiregionali consentono di ottenere una maggiore disponibilità e funzionalità di disaster recovery distribuendo il database su più aree. Regioni AWS

Scegliete il tipo di distribuzione in linea con i requisiti dell'applicazione in termini di disponibilità, prestazioni e distribuzione geografica.

Argomenti

- Creazione di un cluster
- Crea un cluster

- Aggiornamento di un cluster
- Eliminazione di un cluster

Creazione di un cluster

Consulta le seguenti informazioni per imparare a creare cluster a regione singola e multiarea in Aurora DSQL.

Python

Per creare un cluster in un unico cluster Regione AWS, usa 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()
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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={
```

C++

L'esempio seguente consente di creare un cluster in un unico 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;
}
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
#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);

    // We can only set the witness region for the first cluster
    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;

    // For the second cluster we can set witness region and designate cluster1 as a
peer
    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 in un unico cluster Regione AWS, usa 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();
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 updateClusterCommand1 = new UpdateClusterCommand(
            {
                identifier: response1.identifier,
                multiRegionProperties: {
                    witnessRegion: witnessRegion,
                    clusters: [response2.arn]
                }
            }
        );

        await client1.send(updateClusterCommand1);
        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 1 ${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 2 ${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

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

```
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);
        }
    }
}
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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

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

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};
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_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;
    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust single region cluster"),
    )]);

    let create_cluster_output = client
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
```

```
        .send()
        .await
        .unwrap();
    println!("Created {}", create_cluster_output.arn);

    println!("Waiting for cluster to become ACTIVE");
    client
        .wait_until_cluster_active()
        .identifier(&create_cluster_output.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";
    let output = create_cluster(region).await;
    println!("{:#?}", output);
    Ok(())
}
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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

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

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

def create_cluster(region)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.create_cluster(
    deletion_protection_enabled: true,
    tags: {
      Name: "ruby single region cluster"
    }
```

```
  )
  puts "Created #{cluster.arn}"

  # The DSQL SDK offers built-in waiters to poll for a cluster's
  # transition to ACTIVE.
  puts "Waiting for cluster to become ACTIVE"
  client.wait_until(:cluster_active, 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 create cluster in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster = create_cluster(region)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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__
```

.NET

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

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

namespace DSQLExamples.examples
{
    public class CreateSingleRegionCluster
    {
        /// <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 a cluster without delete protection and a name.
        /// </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
                };

                CreateClusterResponse response = await
client.CreateClusterAsync(createClusterRequest);
                Console.WriteLine($"Initiated creation of {response.Arn}");

                return response;
            }
        }

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

            await Create(region);
```

```
        }
    }
}
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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}");
        }
    }
}
```

Golang

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

```
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 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)
 }

 // Create a DSQL client
 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("error creating cluster: %w", err)
 }

 fmt.Printf("Created cluster: %s\n", *clusterProperties.Arn)

 // 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
 })
```

```
 id := clusterProperties.Identifier

 // Create the input for the clusterProperties
 getInput := &dsql.GetClusterInput{
  Identifier: id,
 }

 // Wait for the cluster to become active
 fmt.Println("Waiting for cluster to become ACTIVE")
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to become active: %w", err)
 }

 fmt.Printf("Cluster %s is now active\n", *id)
 return nil
}

// Example usage in main function
func main() {

 region := "us-east-1"

 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 if err := CreateCluster(ctx, region); err != nil {
  log.Fatalf("Failed to create cluster: %v", err)
 }
}
```

Per creare un cluster multiregionale, utilizzare l'esempio seguente. La creazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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)
 }

}
```

Crea un cluster

Consulta le seguenti informazioni per scoprire come restituire informazioni a un cluster in Aurora DSQL.

Python

Per ottenere informazioni su un cluster singolo o multiregionale, usa 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 singolo o multiregionale.

```
#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 singolo o multiregionale, 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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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

Consulta le seguenti informazioni per scoprire come aggiornare un cluster in Aurora DSQL. L'aggiornamento di un cluster può richiedere uno o due minuti. Ti consigliamo di attendere qualche istante e poi eseguire get cluster per conoscere lo stato del cluster.

Python

Per aggiornare un cluster singolo o multiregionale, usa 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 singolo o multiregionale.

```
#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 singolo o multiregionale, 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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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 singolo o multiregionale.

```
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

Consulta le seguenti informazioni per scoprire come eliminare un cluster in Aurora DSQL.

Python

Per eliminare un cluster in un unico cluster Regione AWS, usa il seguente esempio.

```
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()
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente.

```
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 in un unico cluster 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;
}
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
#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 in un unico cluster 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();
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 in un unico cluster 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);
        }
    }
}
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 in un unico cluster 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(())
}
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 in un unico cluster Regione AWS, utilizzare l'esempio seguente.

```
equire "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__
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 in un unico cluster 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);
        }
    }
}
```

Per eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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 in un unico cluster 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 eliminare un cluster multiregionale, utilizzare l'esempio seguente. L'eliminazione di un cluster multiregionale potrebbe richiedere del tempo.

```
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)
 }
```

```
}
```

Tutorial

I seguenti tutorial e codice di esempio GitHub consentono di eseguire attività comuni in Aurora DSQL.

- Utilizzo di Benchbase con Aurora DSQL, un ramo dell'utilità di benchmarking open source Benchbase verificata per funzionare con Aurora DSQL.
- Aurora DSQL loader: questo script Python open source semplifica il caricamento dei dati in Aurora DSQL per i tuoi casi d'uso, come la compilazione di tabelle per il test o il trasferimento di dati in Aurora DSQL.
- Esempi di Aurora DSQL: il `aws-samples/aurora-dsql-samples` repository on GitHub contiene esempi di codice su come connettersi e utilizzare Aurora DSQL in vari linguaggi di programmazione utilizzando i mapper relazionali a oggetti () e AWS SDKs i framework web. ORMs Gli esempi mostrano come eseguire attività comuni, come installare client, gestire l'autenticazione ed eseguire operazioni CRUD.

Utilizzo AWS Lambda con Amazon Aurora DSQL

Il seguente tutorial descrive come usare Lambda con Aurora DSQL

Prerequisiti

- Autorizzazione a creare funzioni Lambda. Per ulteriori informazioni, consulta Guida introduttiva a Lambda.
- Autorizzazione a creare o modificare la policy IAM creata da Lambda. Sono necessarie autorizzazioni `iam:CreatePolicy` e. `iam:AttachRolePolicy` Per ulteriori informazioni, consulta Azioni, risorse e chiavi di condizione per IAM.
- È necessario aver installato npm v8.5.3 o versione successiva.
- Devi aver installato zip v3.0 o versione successiva.

Crea una nuova funzione in AWS Lambda.

1. Accedi a AWS Management Console e apri la AWS Lambda console all'indirizzo https://console.aws.amazon.com/lambda/.
2. Scegli Crea funzione.

3. Fornisci un nome, ad esempio`dsql-sample`.
4. Non modificare le impostazioni predefinite per assicurarti che Lambda crei un nuovo ruolo con autorizzazioni Lambda di base.
5. Scegli Crea funzione.

Autorizza il tuo ruolo di esecuzione Lambda a connettersi al cluster

1. Nella tua funzione Lambda, scegli Configurazione > Autorizzazioni.
2. Scegli il nome del ruolo per aprire il ruolo di esecuzione nella console IAM.
3. Scegli Aggiungi autorizzazioni > Crea policy in linea e usa l'editor JSON.
4. In Action incolla la seguente azione per autorizzare la tua identità IAM a connettersi utilizzando il ruolo del database di amministrazione.

```
"Action": ["dsql:DbConnectAdmin"],
```

> **Note**
>
> Stiamo utilizzando un ruolo di amministratore per ridurre al minimo i passaggi preliminari necessari per iniziare. Non dovresti usare un ruolo di amministratore del database per le tue applicazioni di produzione. Scopri come creare ruoli di database personalizzati con autorizzazione e con il minor numero di autorizzazioni per accedere al database. Utilizzo dei ruoli del database e dell'autenticazione IAM

5. In Resource, aggiungi l'Amazon Resource Name (ARN) del tuo cluster. Puoi anche usare un jolly.

```
"Resource": ["*"]
```

6. Scegli Next (Successivo).
7. Inserisci un nome per la politica, ad esempio`dsql-sample-dbconnect`.
8. Scegliere Create Policy (Crea policy).

Crea un pacchetto da caricare su Lambda.

1. Crea una cartella denominata`myfunction`.

2. Nella cartella, crea un nuovo file denominato `package.json` con il seguente contenuto.

```
{
  "dependencies": {
    "@aws-sdk/dsql-signer": "^3.705.0",
    "assert": "2.1.0",
    "pg": "^8.13.1"
  }
}
```

3. Nella cartella, create un file denominato `index.mjs` nella directory con il seguente contenuto.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";
import pg from "pg";
import assert from "node:assert";
const { Client } = pg;


async function dsql_sample(clusterEndpoint, region) {
  let client;
  try {
    // The token expiration time is optional, and the default value 900 seconds
    const signer = new DsqlSigner({
      hostname: clusterEndpoint,
      region,
    });
    const token = await signer.getDbConnectAdminAuthToken();
    // <https://node-postgres.com/apis/client>
    // By default `rejectUnauthorized` is true in TLS options
    // <https://nodejs.org/api/tls.html#tls_tls_connect_options_callback>
    // The config does not offer any specific parameter to set sslmode to verify-
full
    // Settings are controlled either via connection string or by setting
    // rejectUnauthorized to false in ssl options
    client = new Client({
      host: clusterEndpoint,
      user: "admin",
      password: token,
      database: "postgres",
      port: 5432,
      // <https://node-postgres.com/announcements> for version 8.0
      ssl: true,
      rejectUnauthorized: false
```

```
    });

    // Connect
    await client.connect();

    // Create a new table
    await client.query(`CREATE TABLE IF NOT EXISTS owner (
      id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
      name VARCHAR(30) NOT NULL,
      city VARCHAR(80) NOT NULL,
      telephone VARCHAR(20)
    )`);

    // Insert some data
    await client.query("INSERT INTO owner(name, city, telephone) VALUES($1, $2,
 $3)",
      ["John Doe", "Anytown", "555-555-1900"]
    );

    // Check that data is inserted by reading it back
    const result = await client.query("SELECT id, city FROM owner where name='John
 Doe'");
    assert.deepEqual(result.rows[0].city, "Anytown")
    assert.notEqual(result.rows[0].id, null)

    await client.query("DELETE FROM owner where name='John Doe'");

  } catch (error) {
    console.error(error);
    throw new Error("Failed to connect to the database");
  } finally {
    client?.end();
  }
  Promise.resolve();
}


// https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html
export const handler = async (event) => {
  const endpoint = event.endpoint;
  const region = event.region;
  const responseCode = await dsql_sample(endpoint, region);

  const response = {
```

```
    statusCode: responseCode,
    endpoint: endpoint,
  };
  return response;
};
```

4. Utilizzate i seguenti comandi per creare un pacchetto.

```
npm install
zip -r pkg.zip .
```

Carica il pacchetto di codice e testa la tua funzione Lambda

1. Nella scheda Codice della funzione Lambda, scegli Carica da > .zip file
2. Carica il file che `pkg.zip` hai creato. Per ulteriori informazioni, consulta Distribuire le funzioni Lambda di Node.js con archivi di file.zip.
3. Nella scheda Test della funzione Lambda, incolla il seguente payload JSON e modificalo per utilizzare l'ID del cluster.
4. Nella scheda Test della funzione Lambda, usa il seguente Event JSON modificato per specificare l'endpoint del cluster.

```
{"endpoint": "replace_with_your_cluster_endpoint"}
```

5. Inserisci il nome di un evento, ad esempio. `dsql-sample-test` Scegli Save (Salva).
6. Scegli Test (Esegui test).
7. Scegliete Dettagli per espandere la risposta di esecuzione e l'output del registro.
8. Se ha avuto successo, la risposta di esecuzione della funzione Lambda dovrebbe restituire un codice di stato 200.

```
{"statusCode": 200, "endpoint": "your_cluster_endpoint"}
```

Se il database restituisce un errore o se la connessione al database fallisce, la risposta di esecuzione della funzione Lambda restituisce un codice di stato 500.

```
{"statusCode": 500,"endpoint": "your_cluster_endpoint"}
```

Backup e ripristino per Amazon Aurora DSQL

Amazon Aurora DSQL ti aiuta a soddisfare i requisiti di conformità normativa e 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 tra AWS i servizi, nel cloud e in locale. Il servizio semplifica la creazione, la gestione e il ripristino dei backup per i cluster Aurora DSQL a regione singola e multiregione.

Le caratteristiche principali comprendono:

- Gestione centralizzata dei backup tramite SDK o AWS Management Console AWS CLI
- Backup completi del cluster
- Pianificazioni di backup e politiche di conservazione automatizzate
- Funzionalità interregionali e interaccount
- Configurazione WORM (write-once, read-many) per tutti i backup archiviati

Per ulteriori informazioni sulle funzionalità di AWS Backup Vault Lock e un elenco completo delle AWS Backup funzionalità disponibili per Aurora DSQL, consulta i vantaggi AWS Backup e la disponibilità delle funzionalità di Vault Lock nella Developer Guide.AWS Backup

Guida introduttiva con AWS Backup

AWS Backup crea copie complete dei tuoi cluster Aurora DSQL. Puoi iniziare a usare AWS Backup per Aurora DSQL seguendo la procedura descritta in Guida introduttiva a: AWS Backup

1. Crea backup su richiesta per una protezione immediata.
2. Stabilisci piani di backup per backup automatizzati e pianificati.
3. Configura i periodi di conservazione e la copia tra regioni.
4. Configura il monitoraggio e le notifiche per le attività di backup.

Ripristino dei backup

Quando ripristini i cluster Aurora DSQL, crea AWS Backup sempre nuovi cluster per preservare i dati di origine. Per ripristinare un cluster Aurora DSQL Single-Region, usa /backup https://

console.aws.amazon.como CLI per selezionare il punto di ripristino (backup) che desideri ripristinare. Configura le impostazioni per il nuovo cluster che verrà creato dal backup.

Il ripristino di un cluster multiregionale Aurora DSQL è supportato solo tramite. AWS CLIPer ripristinare un cluster multiregionale Aurora DSQL, è necessario utilizzare sia la CLI Aurora DSQL che la AWS Backup CLI.

Per ripristinare un cluster multiregionale Aurora DSQL.

1. Seleziona il punto di ripristino del tuo cluster multiregionale.
2. Copia il punto di ripristino su un altro Regione AWS che supporti i cluster multiregionali.

   > **Note**
   >
   > Le regioni che non supportano i cluster multiregionali comporteranno un'operazione di ripristino non riuscita.

3. Avvia un processo di ripristino per ogni cluster utilizzando la AWS Backup CLI.
4. Utilizza la Configurazione di cluster multiregionali documentazione per eseguire il peering dei cluster Aurora DSQL appena creati.

Per istruzioni dettagliate su questi passaggi, consulta la documentazione di ripristino SQL di Amazon Aurora.

Per eseguire il ripristino in un cluster Aurora DSQL multiregionale, è possibile utilizzare un backup eseguito in un unico cluster. Regione AWS Tuttavia, prima di iniziare il processo di ripristino, è necessario copiare il backup su un altro Regione AWS che supporti i cluster multiregionali. Questo passaggio garantisce che l'operazione di ripristino possa essere completata correttamente. Ti consigliamo di creare copie di backup in aree chiave Regioni AWS come Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio) o Stati Uniti occidentali (Oregon) per abilitare solide opzioni di disaster recovery e soddisfare i requisiti di conformità.

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.

Vedi Registrazione delle operazioni SQL di Aurora utilizzando AWS CloudTrail per saperne di più sulla registrazione dei record delle azioni intraprese da un utente, un ruolo o Servizio AWS durante l'utilizzo di Aurora DSQL.

Risorse aggiuntive

Per saperne di più sulle AWS Backup funzionalità e sul suo utilizzo in combinazione con Aurora DSQL, consulta le seguenti risorse:

- Politiche gestite per AWS Backup
- Ripristino SQL di Amazon Aurora
- 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 gestiate un singolo cluster o una distribuzione complessa in più regioni, AWS Backup fornisce gli strumenti necessari per garantire che i dati rimangano sicuri e recuperabili.

Monitoraggio e registrazione per Aurora DSQL

Il monitoraggio e la registrazione sono elementi importanti per mantenere l'affidabilità, la disponibilità e le prestazioni delle risorse SQL di Amazon Aurora. È necessario monitorare e raccogliere i dati di registrazione da tutte le parti delle risorse Aurora DSQL in modo da poter eseguire facilmente il debug di un errore multipunto.

- Amazon CloudWatch monitora AWS le tue risorse e le applicazioni su cui esegui AWS in tempo reale. Puoi raccogliere i parametri e tenerne traccia, 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.

Visualizzazione 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' AWS Management Console API SQL di Aurora o Aurora. AWS CLI

Stati e definizioni dei cluster Aurora DSQL

La tabella seguente descrive ogni stato possibile per un cluster Aurora DSQL e il significato di ogni stato.

Stato	Descrizione
Creating (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.

Stato	Descrizione
Active (Attivo)	Il cluster è operativo e pronto all'uso.
Idle	Un cluster diventa inattivo quando rimane inattivo abbastanza a lungo da consentire ad Aurora DSQL di recuperare le risorse configurate per esso. Quando ci si connette a un cluster inattivo, Aurora DSQL riporta il cluster allo stato Attivo.
Inattivo	Un cluster diventa inattivo quando non vi è stata alcuna attività sul cluster per un periodo prolungato. Quando si tenta di connettersi a un cluster inattivo, Aurora DSQL riporta automaticamente il cluster allo stato Attivo.
Aggiornamento in corso	Un cluster passa allo stato di aggiornamento quando si apportano modifiche alla configurazione del cluster.
Deleting (Eliminazione in corso)	Un cluster passa allo stato Eliminazione quando si invia una richiesta di eliminazione.
Deleted (Eliminato)	Il cluster è stato eliminato con successo.
Failed (Non riuscito)	Aurora DSQL non è riuscita a creare il cluster perché ha rilevato un errore.
Configurazione in sospeso	Solo per cluster multiregionali. Un cluster multiregionale passa allo stato di configurazione in sospeso quando si crea un cluster multiregionale nella prima regione con una regione di riferimento. 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.
Eliminazione in sospeso	Solo per cluster multiregionali. Un cluster con più regioni passa allo stato In attesa di eliminazione quando si elimina un cluster da esso. Il cluster passa allo stato di eliminazione dopo l'eliminazione dell'ultimo cluster peer.

Visualizzazione dello stato del cluster Aurora DSQL

Per visualizzare lo stato del cluster, usa l'API SQL AWS Management Console di Aurora o Aurora. AWS CLI

Console

Segui questi passaggi per visualizzare lo stato del cluster in: AWS Management Console

Per visualizzare lo stato del cluster nella console

1. Apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql
2. Nel pannello di navigazione, scegliere Cluster.
3. Visualizza lo stato di ogni cluster nella dashboard.

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
```

Esegui il comando seguente per elencare lo stato di tutti i cluster.

```
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 in fase di eliminazione.

```
aaabbb2bkx555xa7p42qd5cdef    ACTIVE
abcde123efghi77t35abcdefgh    ACTIVE
12abc6lqasc5bbbbbbbbbbbbbb    DELETING
```

Monitoraggio di Aurora DSQL con Amazon CloudWatch

Monitora utilizzando Aurora DSQL CloudWatch, che raccoglie dati grezzi e li elabora in metriche leggibili quasi in tempo reale. CloudWatch conserva queste statistiche per 15 mesi, aiutandoti a

ottenere una prospettiva migliore sulle prestazioni delle tue applicazioni web o dei tuoi servizi. Imposta allarmi per controllare soglie specifiche e inviare notifiche o intraprendere azioni quando vengono raggiunte. Esamina le seguenti metriche di utilizzo e osservabilità disponibili per Aurora DSQL.

Per ulteriori informazioni, consulta la Amazon CloudWatch User Guide.

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.

CloudWatch Nome parametro	Parametro	Unità	Descrizione
ReadOnlyTransactions	Transazioni di sola lettura	nessuno	Il numero di transazioni di sola lettura
TotalTransactions	Transazioni totali	nessuno	Il numero totale di transazioni eseguite sul sistema, incluse le transazioni di sola lettura.
QueryTimeouts	Timeout per query	nessuno	Il numero di interrogazioni scadute a causa del raggiungimento del tempo massimo di transazione
OccConflicts	Conflitti OCC	nessuno	Il numero di transazioni interrotte a causa dell'OCC di livello chiave

CloudWatch Nome parametro	Parametro	Unità	Descrizione
CommitLatency	Latenza del commit	millisecondi	Tempo impiegato nella fase di commit dell'esecuzione delle query (P50)
BytesWritten	Byte scritti	bytes	Byte scritti nella memoria
BytesRead	Byte letti	bytes	Byte letti dalla memoria
ComputeTime	Tempo di calcolo QP	millisecondi	Orologio da parete QP
ClusterStorageSize	Dimensioni di archiviazione del cluster	bytes	Dimensione del cluster

Parametri di utilizzo

Aurora DSQL misura tutte le attività basate sulla richiesta, come l'elaborazione delle query, le letture e le scritture, utilizzando un'unica unità di fatturazione normalizzata denominata Distributed Processing Unit (DPU).

CloudWatch Nome parametro	Parametro	Dimensione: ResourceId	Unità	Descrizione
PU scritta	Unità di scrittura	<cluster-id>	DPU	Approssimativa del componente write-active-use dell'utilizzo della DPU del cluster Aurora DSQL.

CloudWatch Nome parametro	Parametro	Dimensione: ResourceId	Unità	Descrizione
MultiRegionWriteDPU	Unità di scrittura multiregione	<cluster-id>	DPU	Applicabile ai cluster multiregione: approssima il componente di scrittura multi-regione per uso attivo dell'utilizzo della DPU del cluster Aurora DSQL.
Leggi DPU	Unità di lettura	<cluster-id>	DPU	Approssimativa del componente di lettura e uso attivo dell'utilizzo della DPU del cluster Aurora DSQL.
PU calcolata	Unità di calcolo	<cluster-id>	DPU	Approssimativa del componente di calcolo ad uso attivo dell'utilizzo della DPU del cluster Aurora DSQL.
DPU totale	Unità totali	<cluster-id>	DPU	Approssimativa del componente totale ad uso attivo dell'utilizzo della DPU del cluster Aurora DSQL.

Registrazione delle operazioni SQL di Aurora utilizzando AWS CloudTrail

Amazon Aurora DSQL è integrato con AWS CloudTrail, un servizio che fornisce un registro delle azioni intraprese da un utente, ruolo o un. Servizio AWS Esistono due tipi di eventi CloudTrail: eventi di gestione ed eventi relativi ai dati. Gli eventi di gestione vengono emessi per controllare le modifiche alla configurazione AWS delle risorse. Gli eventi relativi ai dati registrano l'utilizzo AWS delle risorse in genere nel piano dati del servizio.

CloudTrail acquisisce tutte le chiamate API per 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 relativi ai dati.

Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata ad Aurora DSQL, l'indirizzo IP da cui è stata effettuata la richiesta, quando è stata effettuata, l'identità dell'utente che ha effettuato la richiesta e dettagli aggiuntivi.

CloudTrail è abilitato per impostazione predefinita nel Account AWS momento in cui si crea l'account e si ha accesso alla cronologia degli CloudTrail eventi. La cronologia CloudTrail degli eventi fornisce un record visualizzabile, ricercabile, scaricabile e immutabile degli ultimi 90 giorni degli eventi di gestione registrati in un. Regione AWSPer ulteriori informazioni, consulta Lavorare con la cronologia degli CloudTrail eventi nella Guida per l'utente.AWS CloudTrail Non sono CloudTrail previsti costi per la registrazione della cronologia degli eventi.

Per creare una registrazione continua degli eventi nel tuo AWS account, inclusi gli eventi per Aurora DSQL, crea un trail o un archivio dati di eventi AWS CloudTrail Lake (una soluzione centralizzata di archiviazione e analisi per gli eventi). AWS CloudTrail Per ulteriori informazioni sulla creazione di percorsi, consulta Lavorare con i percorsi. CloudTrail Per ulteriori informazioni sulla configurazione e la gestione degli archivi dati degli eventi, consulta CloudTrail Lake Event Data Store.

Eventi di gestione Aurora DSQL in CloudTrail

CloudTrail Gli eventi di gestione forniscono informazioni sulle operazioni di gestione eseguite sulle risorse dell'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 tutte le operazioni del piano di controllo Aurora DSQL come eventi di gestione. Per un elenco delle operazioni del piano di controllo DSQL di Amazon Aurora a cui Aurora DSQL accede CloudTrail, consulta il riferimento all'API Aurora DSQL.

Registri del piano di controllo

Amazon Aurora DSQL registra le seguenti operazioni del piano di controllo Aurora DSQL come eventi di gestione. CloudTrail

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName
- ListClusters
- ListTagsForResource
- TagResource
- UntagResource
- UpdateCluster

Registri di backup e ripristino

Amazon Aurora DSQL registra le seguenti operazioni di backup e ripristino di Aurora DSQL come eventi di gestione. CloudTrail

- `StartBackupJob`
- `StopBackupJob`
- `GetBackupJob`
- `StartRestoreJob`
- `StopRestoreJob`
- `GetRestoreJob`

Per ulteriori informazioni sulla protezione dei cluster Aurora DSQL, consulta. AWS BackupBackup e ripristino per Amazon Aurora DSQL

Log AWS KMS

Amazon Aurora DSQL registra le seguenti AWS KMS operazioni come eventi di CloudTrail gestione.

- `GenerateDataKey`
- `Decrypt`

Per ulteriori informazioni su come CloudTrail i log tengono traccia delle richieste inviate da Aurora DSQL AWS KMS per tuo conto, consulta. Monitoraggio dell'interazione di Aurora DSQL con AWS KMS

Eventi relativi ai dati Aurora DSQL in CloudTrail

CloudTrail Gli eventi relativi ai dati in genere forniscono informazioni sulle operazioni eseguite sulle risorse su o all'interno di una risorsa. Questi vengono utilizzati anche per acquisire le operazioni del 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 relativi ai dati. La cronologia CloudTrail degli eventi non registra gli eventi relativi ai dati.

Per ulteriori informazioni su come registrare gli eventi di dati, consulta Registrazione di eventi di dati con AWS Management Console e Registrazione di eventi di dati con AWS Command Line Interface nella Guida all'utente AWS CloudTrail .

Per gli eventi di dati sono previsti costi aggiuntivi. Per ulteriori informazioni sui CloudTrail prezzi, consulta la sezione AWS CloudTrail Prezzi.

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 Aurora DSQL per i quali è possibile registrare gli eventi relativi ai dati. La colonna Tipo di risorsa (console) mostra il valore da scegliere dall'elenco Tipo di risorsa sulla CloudTrail console. La colonna del valore resources.type mostra il `resources.type` valore da specificare durante la configurazione dei selettori di eventi avanzati utilizzando o. AWS CLI CloudTrail APIs La CloudTrail colonna Dati APIs registrati mostra le chiamate API registrate per il tipo di risorsa. CloudTrail

Tipo di risorsa (console)	valore resources.type	Dati APIs registrati su CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

È possibile configurare selettori di eventi avanzati per filtrare `eventName` e `resources.ARN` campi per registrare solo gli eventi filtrati. Per ulteriori informazioni su questi campi, consulta AdvancedFieldSelector in Riferimento API AWS CloudTrail .

L'esempio seguente mostra come utilizzare AWS CLI per configurare la ricezione `dsql-data-events-trail` di 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 in Cloud AWS. AWS fornisce inoltre servizi che è possibile utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per informazioni sui programmi di conformità che si applicano ad Amazon Aurora DSQL, consulta AWS Services in Scope by Compliance Program by Compliance Program.
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua azienda e le leggi e normative vigenti.

Questa documentazione aiuta a capire come applicare il modello di responsabilità condivisa quando si utilizza Aurora DSQL. I seguenti argomenti mostrano 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

- 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 per Aurora DSQL

AWS politiche gestite per Amazon Aurora SQL

Una policy AWS gestita è una policy autonoma creata e amministrata da. AWS AWS le 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 Ti consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i tuoi 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. AWS è 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 AWSnella Guida per l'utente di IAM.

AWS politica gestita: AmazonAurora DSQLFull accesso

Puoi collegarti `AmazonAuroraDSQLFullAccess` ai tuoi utenti, gruppi e ruoli.

Questa politica concede autorizzazioni che consentono l'accesso amministrativo completo ad Aurora DSQL. I responsabili con queste autorizzazioni possono creare, eliminare e aggiornare i cluster Aurora DSQL, inclusi i cluster multiregione. Possono aggiungere e rimuovere tag dai cluster. Possono elencare i cluster e visualizzare informazioni sui singoli cluster. Possono vedere i tag allegati ai cluster Aurora DSQL. Possono connettersi al database come qualsiasi utente, incluso l'amministratore. Possono eseguire operazioni di backup e ripristino per i cluster Aurora

DSQL, tra cui avvio, arresto e monitoraggio dei processi di backup e ripristino. La policy include AWS KMS autorizzazioni che consentono di eseguire operazioni sulle chiavi gestite dal cliente utilizzate per la crittografia dei cluster. Possono visualizzare tutte le metriche del tuo account. CloudWatch Dispongono inoltre delle autorizzazioni per creare ruoli collegati al servizio per il `dsql.amazonaws.com` servizio, necessari per la creazione di cluster.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `dsql`—concede ai principali l'accesso completo ad Aurora DSQL.
- `cloudwatch`—concede l'autorizzazione a pubblicare punti dati metrici su Amazon. CloudWatch
- `iam`—concede l'autorizzazione a 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.

Puoi trovare la `AmazonAuroraDSQLFullAccess` policy sulla console IAM e Access nella Managed Policy Reference Guide. AmazonAurora DSQLFull AWS

AWS politica gestita: AmazonAurora DSQLRead OnlyAccess

Puoi collegarti `AmazonAuroraDSQLReadOnlyAccess` ai tuoi utenti, gruppi e ruoli.

Consente l'accesso in lettura ad Aurora DSQL. I responsabili con queste autorizzazioni possono elencare i cluster e visualizzare informazioni sui singoli cluster. Possono vedere i tag associati ai cluster Aurora DSQL. Possono recuperare e visualizzare qualsiasi metrica del tuo account. CloudWatch

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `dsql`— concede autorizzazioni di sola lettura a tutte le risorse in Aurora DSQL.
- `cloudwatch`— concede l'autorizzazione a recuperare quantità in batch di dati metrici ed eseguire calcoli CloudWatch metrici sui dati recuperati

Puoi trovare la `AmazonAuroraDSQLReadOnlyAccess` policy sulla console IAM e AmazonAuroraDSQLReadOnlyAccessnella Managed Policy Reference Guide. AWS

AWS politica gestita: AmazonAurora DSQLConsole FullAccess

Puoi collegarti `AmazonAuroraDSQLConsoleFullAccess` ai tuoi utenti, gruppi e ruoli.

Consente l'accesso amministrativo completo ad Amazon Aurora DSQL tramite. AWS Management Console I responsabili con queste autorizzazioni possono creare, eliminare e aggiornare i cluster Aurora DSQL, inclusi i cluster multiregione, con la console. Possono elencare i cluster e visualizzare informazioni sui singoli cluster. Possono visualizzare i tag su qualsiasi risorsa del tuo account. Possono connettersi al database come qualsiasi utente, incluso l'amministratore. Possono eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio dei processi di backup e ripristino. La policy include AWS KMS autorizzazioni che consentono di eseguire operazioni sulle chiavi gestite dal cliente utilizzate per la crittografia dei cluster. Possono essere AWS CloudShell avviati da. AWS Management Console Possono visualizzare qualsiasi metrica del CloudWatch tuo account. Dispongono inoltre delle autorizzazioni per creare ruoli collegati al servizio per il `dsql.amazonaws.com` servizio, necessari per la creazione di cluster.

Puoi trovare la `AmazonAuroraDSQLConsoleFullAccess` policy sulla console IAM e AmazonAuroraDSQLConsoleFullAccessnella AWS Managed Policy Reference Guide.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `dsql`—concede autorizzazioni amministrative complete a tutte le risorse in Aurora DSQL tramite. AWS Management Console
- `cloudwatch`—concede l'autorizzazione a recuperare quantità batch di dati metrici ed eseguire calcoli CloudWatch 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.

Puoi trovare la `AmazonAuroraDSQLReadOnlyAccess` policy sulla console IAM e AmazonAuroraDSQLReadOnlyAccessnella Managed Policy Reference Guide. AWS

AWS politica gestita: Aurora DSQLService RolePolicy

Non puoi collegare Aurora DSQLService RolePolicy alle tue entità IAM. Questa policy è associata 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 avvisi automatici sulle modifiche a questa pagina, iscriviti al feed RSS nella pagina della cronologia dei documenti Aurora DSQL.

Modifica	Descrizione	Data
AmazonAuroraDSQLFullAccedi all'aggiornamento	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL, tra cui avvio, arresto e monitoraggio dei lavori.	21 maggio 2025

Modifica	Descrizione	Data
	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.	
AmazonAuroraDSQLConsoleFullAccess update	Aggiunge la capacità di eseguire operazioni di backup e ripristino per i cluster Aurora DSQL tramite. AWS Console Home Ciò include l'avvio, l'arresto e il monitoraggio dei lavori. Supporta anche l'utilizzo di chiavi KMS gestite dal cliente per la crittografia e l'avvio dei cluster. AWS CloudShell Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccessUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	21 maggio 2025

Modifica	Descrizione	Data
AmazonAuroraDSQLFullAccesso — Aggiornamento dell'accesso	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 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 AmazonAuroraDSQLReadOnlyAccessUtilizzo 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 multiregione e la connessione endpoint VPC. Le nuove autorizzazioni includono: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccessUtilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

Modifica	Descrizione	Data
AuroraDsqlServiceLinkedRole Policy update	Aggiunge la possibilità di pubblicare metriche nella policy `AWS/AuroraDSQL` e `AWS/Usage CloudWatch` namespace nella 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
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

Il modello di responsabilità condivisa modello di di si applica alla protezione dei dati in. Come descritto in questo modello, è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi 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, si consiglia di proteggere le credenziali e di configurare i singoli utenti con AWS IAM Identity Center o AWS Identity and Access Management. In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti 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 con AWS CloudTrail. Per informazioni sull'utilizzo dei percorsi per acquisire le attività, consulta Lavorare con i percorsi nella Guida per l'utente.
- Utilizza soluzioni di crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno 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.

Ti consigliamo vivamente di non inserire mai informazioni riservate o sensibili, come gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero come il campo Nome. Ciò include quando lavori o utilizzi la console, l'API o. AWS CLI AWS SDKs 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 fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Crittografia dei dati

Amazon Aurora DSQL fornisce un'infrastruttura di storage altamente durevole progettata per lo storage di dati primari e mission-critical. I dati vengono archiviati in modo ridondante su più dispositivi in più strutture in una regione Aurora DSQL.

Crittografia in transito

Per impostazione predefinita, la crittografia in transito è configurata automaticamente. 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 ulteriori informazioni, consulta Processo di firma Signature Version 4 in Riferimenti generali di AWS.

- Usa il AWS CLI o uno dei due per effettuare richieste AWS SDKs a. AWS Questi strumenti firmano automaticamente le tue richieste con la chiave di accesso specificata al momento della configurazione.

Per la crittografia a riposo, vedereCrittografia inattiva in Aurora DSQL.

Riservatezza del traffico Internet

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 ulteriori informazioni, consulta What is AWS Site-to-Site VPN?
- Una AWS Direct Connect connessione. Per ulteriori informazioni, vedi Cos'è AWS Direct Connect?

È possibile accedere ad Aurora DSQL tramite la rete utilizzando AWS le operazioni API pubblicate. 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.

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, se non è già presente.

Si consiglia di utilizzare PostgreSQL versione 17.

> **Important**
>
> Per gli ambienti di produzione, consigliamo di utilizzare la modalità `verify-full` SSL 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, puoi seguire i passaggi seguenti.

Linux () RedHat/CentOS/Fedora

Esegui il seguente comando nel tuo terminale:

```
trust list | grep "Amazon Root CA 1"
```

Se il certificato è installato, viene visualizzato il seguente output:

```
label: Amazon Root CA 1
```

macOS

1. Apri Spotlight Search (Comando+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 Windows psql, l'utilizzo dei certificati root di sistema (`sslrootcert=system`) può restituire il seguente errore:`SSL error: unregistered scheme`. Puoi seguire il Connessione da Windows metodo alternativo per connetterti al tuo cluster tramite SSL.

Se Amazon Root CA 1 non è installato nel tuo sistema operativo, procedi nel seguente modo.

Installazione dei certificati

Se il `Amazon Root CA 1` certificato non è preinstallato sul sistema operativo, sarà necessario installarlo manualmente per stabilire connessioni sicure al cluster Aurora DSQL.

Installazione del certificato Linux

Segui questi passaggi per installare il certificato Amazon Root CA su sistemi Linux.

1. Scarica il certificato root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Copia il certificato nel trust store:

```
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 macOS

Questi passaggi di installazione dei certificati sono facoltativi. Funzionano Installazione del certificato Linux anche per macOS.

1. Scarica il certificato principale:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Aggiungi il certificato al portachiavi 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 CLI configurato 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. Connect 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 ulteriori informazioni sulle impostazioni PGSSLMODE, consulta sslmode nella documentazione sulle funzioni di controllo della connessione del database 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. Connect 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 principale e salvalo con 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. Connect usando 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. Connect 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. Connect al database:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
```

```
--username admin `
--host your-cluster-endpoint
```

Risorse aggiuntive

- Documentazione SSL PostgreSQL
- Servizi Amazon Trust

Crittografia dei dati per Amazon Aurora DSQL

Amazon Aurora DSQL crittografa tutti i dati utente inattivi. 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. Encryption at rest ti aiuta a:

- Riduci l'onere operativo legato alla protezione dei dati sensibili
- Crea applicazioni sensibili alla sicurezza che soddisfino i rigorosi requisiti normativi e di conformità alla crittografia
- Aggiungi un ulteriore livello di protezione dei dati proteggendo sempre i dati in un cluster crittografato
- Rispetta le politiche organizzative, le normative di settore o governative e i requisiti di conformità

Con Aurora DSQL, puoi creare applicazioni sensibili alla sicurezza che soddisfano rigorosi requisiti normativi e di conformità alla 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 chiavi KMS per Aurora DSQL
- Crittografia inattiva in Aurora DSQL
- Utilizzo AWS KMS e chiavi dati con Aurora DSQL
- Autorizzazione all'uso del tuo AWS KMS key per Aurora DSQL
- Contesto di crittografia Aurora DSQL
- Monitoraggio dell'interazione di Aurora DSQL con AWS KMS
- Creazione di un cluster Aurora DSQL crittografato

- Rimozione o aggiornamento di una chiave per il cluster Aurora DSQL
- Considerazioni sulla crittografia con Aurora DSQL

Tipi di chiavi KMS per Aurora DSQL

Aurora DSQL si integra con la gestione delle chiavi di crittografia AWS KMS per i cluster. Per ulteriori informazioni sui tipi e gli stati delle chiavi, consulta AWS Key Management Service i concetti nella Guida per gli sviluppatori.AWS Key Management Service Quando crei un nuovo cluster, puoi scegliere tra i seguenti tipi di chiavi 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 accedi 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

Sei tu a creare, possedere e gestire la chiave nel tuo. Account AWS Hai il pieno controllo sulla chiave KMS. AWS KMS si 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 ulteriori informazioni, consulta la pagina dei prezzi di AWS KMS.

Puoi passare da un tipo di chiave all'altro in qualsiasi momento. Per ulteriori informazioni sui tipi di chiave, consulta Customer managed keys e Chiavi di proprietà di AWSnella AWS Key Management Service Developer Guide.

> **Note**
>
> La crittografia Aurora DSQL at Rest è disponibile in tutte le regioni in AWS cui è disponibile Aurora DSQL.

Crittografia inattiva in Aurora DSQL

Amazon Aurora DSQL utilizza Advanced Encryption Standard (AES-256) a 256 bit per crittografare i dati inattivi. Questa crittografia aiuta a proteggere i dati dall'accesso non autorizzato allo storage sottostante. AWS KMS gestisce le chiavi di crittografia per i cluster. È possibile utilizzare l'impostazione predefinita Chiavi di proprietà di AWS o scegliere di utilizzare la propria. AWS KMS Chiavi gestite dal cliente Per ulteriori informazioni sulla specificazione e la gestione delle chiavi per i cluster Aurora DSQL, consulta e. Creazione di un cluster Aurora DSQL crittografato Rimozione o aggiornamento di una chiave per il cluster Aurora DSQL

Argomenti

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 del tuo 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 Chiavi di proprietà di AWSin merito Chiavi di proprietà di AWS, consulta la Guida per AWS Key Management Service gli sviluppatori.

Chiavi gestite dal cliente

Puoi creare, possedere e gestire le chiavi gestite dai clienti nel tuo Account AWS. Hai il pieno controllo su queste chiavi KMS, comprese le relative politiche, il materiale di crittografia, i tag e gli alias. Per ulteriori informazioni sulla gestione delle autorizzazioni, consulta Customer managed keys nella Developer Guide.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 disattiva la chiave gestita dal cliente, si pianifica l'eliminazione della chiave o si utilizza una politica che limita l'accesso al servizio, lo stato di crittografia del cluster cambia a. `KMS_KEY_INACCESSIBLE` Quando Aurora DSQL non può accedere alla chiave, gli utenti non possono connettersi al cluster, lo stato di crittografia del cluster cambia e il servizio perde l'accesso ai dati del cluster. `KMS_KEY_INACCESSIBLE`

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 multiregionale, 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 le normali operazioni.

> **Note**
>
> Se Aurora DSQL non riesce ad accedere alla chiave gestita dal cliente, lo stato di crittografia del cluster cambia in. `KMS_KEY_INACCESSIBLE` Dopo aver ripristinato l'accesso alla chiave, il servizio rileverà automaticamente il ripristino entro 15 minuti. Per ulteriori informazioni, vedere Cluster idling.
> Per i cluster multiregionali, 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.

Ti consigliamo di pianificare la tua strategia di crittografia prima di implementare il cluster in Aurora DSQL. Se memorizzi dati sensibili o riservati in Aurora DSQL, prendi in considerazione l'inclusione della crittografia lato client nel tuo piano. In questo modo puoi crittografare i dati il più vicino possibile alla loro origine e garantirne la protezione per tutto il loro ciclo di vita.

Argomenti

- Usare AWS KMS key s con Aurora DSQL
- Utilizzo delle chiavi del cluster con Aurora DSQL
- Memorizzazione nella cache delle chiavi del cluster

Usare AWS KMS key s con Aurora DSQL

Encryption at rest protegge il cluster Aurora DSQL in un. 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. Ma puoi crittografare i tuoi cluster Aurora DSQL con

una chiave gestita dal cliente nel tuo. Account AWS Puoi selezionare una chiave KMS diversa per ogni cluster, anche se fa parte di una configurazione multiregionale.

Si seleziona la chiave KMS per un cluster quando si crea o si aggiorna il cluster. È possibile modificare la chiave KMS per un cluster in qualsiasi momento, nella console Aurora DSQL o utilizzando l'operazione. `UpdateCluster` Il processo di cambio delle chiavi non richiede tempi di inattività o peggioramento 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 offre i seguenti vantaggi.

- Puoi creare e gestire la chiave KMS, inclusa l'impostazione delle politiche chiave e delle politiche 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.
- Puoi 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 sei 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 comportano un addebito per ogni chiamata API e a queste chiavi si applicano delle AWS KMS quote.

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 unica per ogni struttura sottostante in un

cluster, ma più elementi del cluster potrebbero essere protetti 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 di decrittografia 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, memorizza le chiavi crittografate con i dati crittografati in modo che siano disponibili per decrittografare i dati del cluster su richiesta.

Se si modifica la chiave KMS per il cluster, Aurora DSQL crittografa nuovamente la chiave del cluster esistente con la nuova chiave KMS.

Memorizzazione nella cache delle chiavi 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 utilizzi una chiave gestita dal cliente nel tuo account per proteggere il tuo cluster Aurora DSQL, le policy su quella chiave devono autorizzare Aurora DSQL a utilizzarla per tuo conto.

Hai il pieno controllo sulle politiche 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 delle chiavi 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 chiave o in una policy IAM.

Come minimo, Aurora DSQL richiede le seguenti autorizzazioni su una chiave gestita dal cliente:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(per e) kms: ReEncryptFrom kms: ReEncryptTo
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Ad esempio, la policy di 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 i principali specificati nell'informativa non dispongono dell'autorizzazione per utilizzare Aurora DSQL, la chiamata ha esito negativo, anche quando proviene dal servizio Aurora DSQL.
- La chiave `kms:ViaService` condition consente le autorizzazioni solo quando la richiesta proviene da Aurora DSQL per conto dei principali elencati nell'informativa. Tali entità non possono chiamare direttamente queste operazioni.
- Fornisce agli AWS KMS key amministratori (utenti che possono assumere il `db-team` ruolo) l'accesso in sola lettura a 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 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 in testo semplice nei log come quelli di. AWS CloudTrail

Il contesto di crittografia può essere utilizzato anche come condizione per l'autorizzazione nelle politiche.

Nelle sue richieste a AWS 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 crittografato da Aurora DSQL. La chiave è `aws:dsql:ClusterId`. Il valore è l'identificatore 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 abiliti la crittografia a riposo su un cluster, Aurora DSQL crea una chiave 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 del servizio Aurora DSQL. I parametri includono l'Amazon Resource Name (ARN) di AWS 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"
}
```


Decrypt

Quando si accede a un cluster Aurora DSQL crittografato, Aurora DSQL deve decrittografare la chiave del cluster in modo da poter decrittografare le chiavi sottostanti nella gerarchia. Quindi decripta 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 KMS ricava 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 AWS Management Console o da. AWS CLI

Console

Per creare un cluster crittografato in AWS Management Console

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Nel riquadro di navigazione sul lato sinistro della console, scegli Cluster.
3. Scegli Crea cluster in alto a destra e seleziona Single-Region.
4. Nelle impostazioni di crittografia del cluster, scegli una delle seguenti opzioni.
   - Accetta le impostazioni predefinite per crittografare 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 Create cluster (Crea cluster).

Per confermare il tipo di crittografia per il cluster, vai alla pagina Cluster e seleziona l'ID del cluster per visualizzare i dettagli del cluster. 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 di impostare la politica delle chiavi KMS in modo appropriato. Per esempi e ulteriori informazioni, consulta. the section called “Policy delle chiavi per una chiave gestita dal cliente”

CLI

Per creare un cluster crittografato con l'impostazione predefinita Chiave di proprietà di AWS

- Utilizzare 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"
}
```

Per creare un cluster crittografato con la chiave gestita dal cliente

- Usa il comando seguente per creare un cluster Aurora DSQL, sostituendo l'ID della chiave in rosso con l'ID della chiave gestita dal 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 tua chiave.

```
"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 AWS Management Console o the AWS CLI per aggiornare o rimuovere le chiavi di crittografia sui cluster esistenti in Amazon Aurora DSQL. Se si rimuove 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 dal. AWS CLI

Console

Per aggiornare o rimuovere una chiave di crittografia in AWS Management Console

1. Accedi alla console di AWS gestione e apri la console Aurora DSQL all'indirizzo. https://console.aws.amazon.com/dsql/
2. Nel riquadro di navigazione sul lato sinistro della console, scegli Cluster.
3. Dalla visualizzazione a elenco, trova e seleziona la riga del cluster che desideri aggiornare.
4. Seleziona il menu Azioni, 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 una Chiave 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"
```

La descrizione `EncryptionStatus` del cluster è impostata su `ENABLED` e su`AWS_OWNED_KMS_KEY`. `EncryptionType`

```
"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.

Per aggiornare un cluster crittografato con una chiave gestita dal cliente per Aurora DSQL

Aggiornate 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
```

La descrizione `EncryptionStatus` del cluster passa a `UPDATING` e la `EncryptionType` è`CUSTOMER_MANAGED_KMS_KEY`. Dopo che Aurora DSQL avrà terminato la propagazione della nuova chiave attraverso la piattaforma, lo stato di crittografia verrà trasferito a `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 di impostare la politica delle chiavi KMS in modo appropriato. Per esempi e ulteriori informazioni, consulta. the section called “Policy delle chiavi per una chiave gestita dal cliente”

Considerazioni sulla crittografia con Aurora DSQL

- Aurora DSQL crittografa tutti i dati del cluster inattivi. Non è possibile disabilitare questa crittografia o crittografare solo alcuni elementi in un cluster.
- AWS Backup crittografa i tuoi 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.
- I seguenti stati di protezione dei dati sono abilitati per Aurora DSQL:
  - Dati inattivi: Aurora DSQL crittografa tutti i dati statici su supporti di archiviazione persistenti
  - Dati in transito: Aurora DSQL crittografa tutte le comunicazioni utilizzando Transport Layer Security (TLS) per impostazione predefinita
- Quando si passa a una chiave diversa, si consiglia di mantenere abilitata la chiave originale fino al completamento della transizione. AWS necessita della chiave originale per decrittografare i dati prima di crittografare i dati con la nuova chiave. Il processo è completo quando il cluster è attivo `ENABLED` e `encryptionStatus` viene visualizzata la nuova chiave `kmsKeyArn` gestita dal cliente.
- Quando disabiliti la Customer Managed Key o revochi l'accesso ad Aurora DSQL per utilizzare la tua chiave, il cluster entrerà in uno stato. `IDLE`
- L'API SQL di Amazon Aurora AWS Management Console e Amazon Aurora utilizzano termini diversi per i tipi di crittografia:
  - AWS Console: nella console, vedrai `KMS` quando usi una chiave gestita dal cliente e `DEFAULT` quando usi un. Chiave di proprietà di AWS
  - API: l'API SQL di Amazon Aurora viene utilizzata `CUSTOMER_MANAGED_KMS_KEY` per le chiavi gestite dai clienti e per. `AWS_OWNED_KMS_KEY` 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 AWS Management Console AWS 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 (effettuato l'accesso) e autorizzato (dispone delle autorizzazioni) a utilizzare le risorse DSQL di Aurora. IAM è uno strumento Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Argomenti

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia a seconda del lavoro svolto in Aurora DSQL.

Utente del servizio: se utilizzi il servizio Aurora DSQL per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più funzionalità di Aurora DSQL per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non è possibile accedere a una funzionalità di Aurora DSQL, vedere. Risoluzione dei problemi relativi all'identità e all'accesso ad Amazon Aurora SQL

Amministratore del servizio: se sei responsabile delle risorse Aurora DSQL presso la tua azienda, probabilmente hai pieno accesso ad Aurora DSQL. È tuo compito determinare a quali funzionalità e risorse di Aurora DSQL devono accedere gli utenti del servizio. Devi inviare le richieste

all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per ulteriori informazioni su come la tua azienda può utilizzare IAM con Aurora DSQL, consulta. In che modo Amazon Aurora DSQL funziona con IAM

Amministratore IAM: se sei un amministratore IAM, potresti voler conoscere i dettagli su come scrivere policy per gestire l'accesso ad Aurora DSQL. Per visualizzare esempi di policy basate sull'identità di Aurora DSQL che è possibile utilizzare in IAM, vedere. Esempi di policy basate sull'identità per Amazon Aurora DSQL

Autenticazione con identità

L'autenticazione è il modo in cui accedi utilizzando le tue credenziali di identità. AWS Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sul metodo consigliato per la firma delle richieste, consulta Signature Version 4 AWS per le richieste API nella Guida per l'utente IAM.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA)AWS in IAM nella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, di un provider di identità Web AWS Directory Service, della directory Identity Center o di qualsiasi utente che accede utilizzando le Servizi AWS credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni su IAM Identity Center, consulta Cos'è IAM Identity Center? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, se si hanno casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti

alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, potresti avere un gruppo denominato IAMAdminse concedere a quel gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Per assumere temporaneamente un ruolo IAM in AWS Management Console, puoi passare da un ruolo utente a un ruolo IAM (console). Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Create a role for a third-party identity provider (federation) nella Guida per l'utente IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze 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 IAM.

- Accesso a più servizi: alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è normale che quel servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Sessioni di accesso inoltrato (FAS): quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, combinate con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta Forward access sessions.
  - Ruolo di servizio: 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 ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.
  - Ruolo collegato al servizio: 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'azione 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 ai servizi, ma non modificarle.
- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un' EC2 istanza e che AWS CLI effettuano richieste AWS API. È preferibile archiviare le chiavi di accesso all'interno dell' EC2 istanza. Per assegnare un AWS ruolo a un' EC2 istanza e renderlo disponibile per tutte le sue applicazioni, create un profilo di istanza collegato all'istanza. Un profilo di istanza contiene il ruolo e consente ai programmi in esecuzione sull' EC2 istanza di ottenere credenziali temporanee. Per ulteriori informazioni, consulta Utilizzare un ruolo IAM per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon nella IAM User Guide.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni.

AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate sull'identità

Le policy basate su 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 IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente IAM.

Policy basate sulle risorse

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 dei 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. È necessario specificare un principale in una policy basata sulle risorse. 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 puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Elenchi di controllo degli accessi () ACLs

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.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioni ACLs, consulta la panoramica della lista di controllo degli accessi (ACL) nella Amazon Simple Storage Service Developer Guide.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzionalità avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal`sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente IAM.
- Politiche di controllo del servizio (SCPs): SCPs sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più di proprietà dell' Account AWS azienda. Se abiliti tutte le funzionalità di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCPs) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità presenti negli account dei membri, inclusa ciascuna di esse. Utente root dell'account AWS Per ulteriori informazioni su Organizations and SCPs, consulta le politiche di controllo dei servizi nella Guida AWS Organizations per l'utente.

- Politiche di controllo delle risorse (RCPs): RCPs sono politiche JSON che puoi utilizzare per impostare le autorizzazioni massime disponibili per le risorse nei tuoi account senza aggiornare le politiche IAM allegate a ciascuna risorsa di tua proprietà. L'RCP limita le autorizzazioni per le risorse negli account dei membri e può influire sulle autorizzazioni effettive per le identità, incluse le Utente root dell'account AWS, indipendentemente dal fatto che appartengano o meno all'organizzazione. Per ulteriori informazioni su Organizations e RCPs, incluso un elenco di Servizi AWS tale supporto RCPs, vedere Resource control policies (RCPs) nella Guida per l'AWS Organizations utente.
- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, 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 la logica di valutazione delle policy nella IAM User Guide.

In che modo Amazon Aurora DSQL funziona con IAM

Prima di utilizzare IAM per gestire l'accesso ad Aurora DSQL, scopri quali funzionalità IAM sono disponibili per l'uso con Aurora DSQL.

Funzionalità IAM che puoi usare con Amazon Aurora DSQL

Funzionalità IAM	Supporto Aurora DSQL
Policy basate su identità	Sì
Policy basate su risorse	No
Azioni di policy	Sì
Risorse relative alle policy	Sì

Funzionalità IAM	Supporto Aurora DSQL
Chiavi di condizione delle policy	Sì
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, AWS consulta i servizi che funzionano con IAM nella IAM User Guide.

Policy basate sull'identità per Aurora DSQL

Supporta le policy basate su identità: sì

Le policy basate su 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 IAM.

Con le policy basate su 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. Non è possibile specificare l'entità principale in una policy basata sull'identità perché si applica all'utente o al ruolo a cui è associato. 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 di IAM.

Esempi di policy basate sull'identità per Aurora DSQL

Per visualizzare esempi di politiche basate sull'identità di Aurora DSQL, vedere. Esempi di policy basate sull'identità per Amazon Aurora DSQL

Policy basate sulle risorse all'interno di Aurora DSQL

Supporta le policy basate su risorse: no

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 dei 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. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, puoi specificare un intero account o entità IAM in un altro account come principale in una policy basata sulle risorse. L'aggiunta di un principale multi-account a una policy basata sulle risorse rappresenta solo una parte della relazione di trust. Quando il principale e la risorsa sono diversi Account AWS, un amministratore IAM dell'account affidabile deve inoltre concedere all'entità principale (utente o ruolo) l'autorizzazione ad accedere alla risorsa. L'autorizzazione viene concessa collegando all'entità una policy basata sull'identità. Tuttavia, se una policy basata su risorse concede l'accesso a un principale nello stesso account, non sono richieste ulteriori policy basate su identità. Per ulteriori informazioni, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Azioni politiche per Aurora DSQL

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale 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 a un criterio. Le azioni politiche in genere hanno lo stesso nome dell'operazione AWS API associata. Ci sono alcune eccezioni, ad esempio le operazioni di sola autorizzazione che non hanno un'operazione API corrispondente. Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di azioni Aurora DSQL, consulta Actions Defined by Amazon Aurora DSQL nel Service Authorization Reference.

Le azioni politiche in Aurora DSQL utilizzano il seguente prefisso prima dell'azione:

```
dsql
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
      "dsql:action1",
      "dsql:action2"
]
```

Per visualizzare esempi di politiche basate sull'identità di Aurora DSQL, vedere. Esempi di policy basate sull'identità per Amazon Aurora DSQL

Risorse relative alle policy per Aurora DSQL

Supporta le risorse di policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale 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. Le istruzioni devono includere un elemento `Resource`o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). È possibile eseguire questa operazione per operazioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le azioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, 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 sapere con quali azioni è possibile specificare l'ARN di ogni risorsa, consulta Azioni definite da Amazon Aurora DSQL.

Per visualizzare esempi di politiche basate sull'identità di Aurora DSQL, vedere. Esempi di policy basate sull'identità per Amazon Aurora DSQL

Chiavi delle condizioni 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 principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento `Condition`(o blocco `Condition`) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento `Condition`è facoltativo. 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.

Se specifichi più elementi `Condition`in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione `AND`logica. Se si specificano più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione logica. `OR` Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

È possibile anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, è possibile autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta Elementi delle policy IAM: variabili e tag nella Guida per l'utente di IAM.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche del servizio. 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 visualizzare un elenco di chiavi di condizione Aurora DSQL, consulta Chiavi di condizione per Amazon Aurora DSQL nel Service Authorization Reference. Per sapere con quali azioni e risorse puoi utilizzare una chiave di condizione, consulta Azioni definite da Amazon Aurora DSQL.

Per visualizzare esempi di politiche basate sull'identità di Aurora DSQL, vedere. 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 dell'accesso basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi. In AWS, questi attributi sono chiamati tag. Puoi allegare tag a entità IAM (utenti o ruoli) e a molte AWS risorse. L'assegnazione di tag alle entità e alle risorse è il primo passaggio di ABAC. In seguito, vengono progettate policy ABAC per consentire operazioni quando il tag dell'entità principale corrisponde al tag sulla risorsa a cui si sta provando ad accedere.

La strategia ABAC è utile in ambienti soggetti a una rapida crescita e aiuta in situazioni in cui la gestione delle policy diventa impegnativa.

Per controllare l'accesso basato su tag, fornisci 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 Yes (Sì). Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per ulteriori informazioni su ABAC, consulta Definizione delle autorizzazioni con autorizzazione ABAC nella Guida per l'utente 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

Alcune Servizi AWS non funzionano quando si accede utilizzando credenziali temporanee. Per ulteriori informazioni, incluse quelle che Servizi AWS funzionano con credenziali temporanee, consulta la sezione relativa alla Servizi AWS compatibilità con IAM nella IAM User Guide.

Stai utilizzando credenziali temporanee se accedi AWS Management Console utilizzando qualsiasi metodo tranne nome utente e password. Ad esempio, quando accedi AWS utilizzando il link Single Sign-On (SSO) della tua azienda, tale processo crea automaticamente credenziali temporanee. Le credenziali temporanee vengono create in automatico anche quando accedi alla console come utente

e poi cambi ruolo. Per ulteriori informazioni sullo scambio dei ruoli, consulta Passaggio da un ruolo utente a un ruolo IAM (console) nella Guida per l'utente IAM.

È possibile creare manualmente credenziali temporanee utilizzando l'API or. AWS CLI AWS È quindi possibile utilizzare tali credenziali temporanee per accedere. AWS AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza provvisorie in IAM.

Autorizzazioni principali multiservizio per Aurora DSQL

Supporta l'inoltro delle sessioni di accesso (FAS): sì

Quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, in combinazione con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta 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 ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe interrompere la funzionalità di Aurora DSQL. Modifica i ruoli di servizio solo quando Aurora DSQL fornisce indicazioni in tal senso.

Ruoli collegati ai servizi per Aurora DSQL

Supporta 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'azione 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 ai servizi, ma non modificarle.

Per informazioni dettagliate sulla creazione o la gestione di ruoli collegati ai servizi per Aurora DSQL, vedere. Utilizzo di ruoli collegati ai servizi 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 Aurora DSQL. Inoltre, non possono eseguire attività utilizzando AWS Management Console, AWS Command Line Interface (AWS CLI) o AWS l'API. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

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 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 Aurora DSQL
- Consentire agli utenti di visualizzare le loro autorizzazioni

Best practice per le policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare le risorse Aurora DSQL nel tuo account. Queste azioni possono comportare costi aggiuntivi per l' Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Inizia con le policy 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 ulteriori informazioni, consulta Policy gestite da AWSo Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.

- Applica le autorizzazioni con privilegio minimo: quando imposti le autorizzazioni con le policy IAM, concedi 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 privilegi minimi. Per ulteriori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle tue 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 specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente 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, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta Protezione dell'accesso API con MFA nella Guida per l'utente 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 Aurora DSQL

Per accedere alla console DSQL di Amazon Aurora, devi disporre di un set minimo di autorizzazioni. 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 CLI API. AWS Al contrario, concedi 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 ulteriori informazioni, consulta Aggiunta di autorizzazioni a un utente nella Guida per l'utente 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 cpllegate 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 relativi all'identità e all'accesso ad Amazon Aurora SQL

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con Aurora DSQL e IAM.

Argomenti

Non sono autorizzato a eseguire un'azione in Aurora DSQL

Se ricevi un errore che indica che non disponi dell'autorizzazione per eseguire un'operazione, le tue policy devono essere aggiornate in modo che ti sei consentito eseguire tale operazione.

L'errore di esempio seguente si verifica quando `mateojackson` tenta di utilizzare la console per visualizzare i dettagli sulla *`my-dsql-cluster`* risorsa ma non dispone delle *`GetCluster`* autorizzazioni.

```
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, contatta l'amministratore . L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un messaggio di errore indicante che non sei autorizzato a eseguire l'`iam:PassRole`azione, le tue politiche devono essere aggiornate per consentirti di passare 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` tenta di utilizzare la console per eseguire un'azione 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 passare 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 ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se Aurora DSQL supporta queste funzionalità, consulta. In che modo Amazon Aurora DSQL funziona con IAM
- Per scoprire come fornire l'accesso alle risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM di un altro Account AWS utente di tua proprietà nella IAM User Guide. Account AWS
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account 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 IAM.

Utilizzo di ruoli collegati ai servizi in Aurora DSQL

Aurora DSQL utilizza ruoli collegati ai servizi AWS Identity and Access Management (IAM). Un ruolo collegato ai servizi è un tipo unico di ruolo IAM 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 ai servizi semplificano il processo di configurazione perché non è necessario aggiungere manualmente le autorizzazioni necessarie per utilizzare Aurora DSQL. Quando crei un cluster, Aurora DSQL crea automaticamente un ruolo collegato al servizio per te. È possibile eliminare il ruolo collegato al servizio solo dopo aver eliminato tutti i cluster. Ciò protegge le risorse Aurora DSQL perché non è possibile rimuovere inavvertitamente le autorizzazioni necessarie per l'accesso alle risorse.

Per informazioni sugli altri servizi che supportano i ruoli collegati ai servizi, consulta Servizi AWS che funzionano con IAM e cerca i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegli Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

I ruoli collegati ai servizi sono disponibili in tutte le regioni Aurora DSQL supportate.

Autorizzazioni di ruolo collegate ai servizi per Aurora SQL

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 ai servizi devi configurare le relative autorizzazioni. Potresti

riscontrare il seguente messaggio di errore:. `You don't have the permissions to create an Amazon Aurora DSQL service-linked role` Se viene visualizzato questo messaggio, assicurati che le autorizzazioni seguenti siano abilitate:

```
{
"Sid" : "CreateDsqlServiceLinkedRole",
    "Effect" : "Allow",
    "Action" : "iam:CreateServiceLinkedRole",
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "iam:AWSServiceName" : "dsql.amazonaws.com"
      }
    }
  }
```

Per ulteriori informazioni, vedere Autorizzazioni dei ruoli collegati ai servizi.

Creare un ruolo collegato ai servizi

Non è necessario creare manualmente un ruolo collegato ai DSQLService LinkedRolePolicy servizi Aurora. Aurora DSQL crea il ruolo collegato al servizio per te. 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 ai servizi

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 ai servizi

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, ti consigliamo di eliminare il ruolo. In questo modo, non hai un'entità inutilizzata che non viene monitorata o gestita attivamente.

Prima di poter eliminare un ruolo collegato al servizio per un account, è necessario eliminare tutti i cluster presenti nell'account.

Puoi utilizzare la console IAM AWS CLI, o l'API IAM per eliminare un ruolo collegato al servizio. Per ulteriori informazioni, consulta Create a service-linked role nella IAM User Guide.

Regioni supportate per i ruoli collegati ai servizi Aurora DSQL

Aurora DSQL supporta l'utilizzo di ruoli collegati ai servizi in tutte le regioni in cui il servizio è disponibile. Per ulteriori informazioni, consulta AWS Regioni ed endpoint.

Utilizzo delle chiavi di condizione IAM con Amazon Aurora DSQL

Quando si concedono le autorizzazioni in Aurora DSQL, è possibile specificare le condizioni che determinano l'effetto di una politica di autorizzazioni. Di seguito sono riportati alcuni esempi di come è possibile utilizzare le chiavi di condizione nelle politiche di autorizzazione di Aurora DSQL.

Esempio 1: concedere l'autorizzazione per creare un cluster in uno specifico Regione AWS

La seguente politica concede l'autorizzazione a creare cluster nelle regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti orientali (Ohio). Questa policy utilizza la risorsa ARN per limitare le regioni consentite, quindi Aurora DSQL può creare cluster solo se tale ARN è specificato nella sezione della policy. `Resource`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            # Control where clusters can be created
            "Action": ["CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Esempio 2: concedere l'autorizzazione per creare un cluster multiregionale in s specifici Regione AWS

La seguente politica concede l'autorizzazione a creare cluster multiregionali nelle regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti orientali (Ohio). Questa politica utilizza la risorsa ARN per limitare le regioni consentite, quindi Aurora DSQL può creare cluster multiregionali solo se questo ARN è specificato nella sezione della politica. `Resource` Tieni presente che la creazione di cluster multiregionali richiede anche le`PutMultiRegionProperties`, `PutWitnessRegion` e le autorizzazioni in ciascuna regione specificata. `AddPeerCluster`

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: concedere l'autorizzazione per creare un cluster multiregionale con una regione di riferimento specifica

La seguente politica utilizza una chiave di `dsql:WitnessRegion` condizione Aurora DSQL e consente a un utente di creare cluster multiregionali con una regione di riferimento negli Stati Uniti occidentali (Oregon). Se non si specifica la `dsql:WitnessRegion` condizione, è possibile utilizzare qualsiasi regione come regione di riferimento.

```
{
    "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

La sicurezza è la massima priorità in. AWS 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. AWS è responsabile di qualsiasi risposta agli incidenti relativi al servizio SQL di Amazon Aurora. Inoltre, in qualità di 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, dal punto di vista del modello di responsabilità condivisa, sei responsabile della risposta agli incidenti.

Stabilendo una linea di base di sicurezza che soddisfi gli obiettivi delle applicazioni eseguite nel cloud, sei in grado di rilevare deviazioni a cui puoi rispondere. Per aiutarvi a comprendere l'impatto che la risposta agli incidenti e le vostre scelte hanno sugli obiettivi aziendali, vi invitiamo a consultare le seguenti risorse:

- AWS Guida alla risposta agli incidenti di sicurezza
- AWS Le 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 SQL

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 AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS 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. AWS fornisce le seguenti risorse per contribuire alla conformità:

- Governance e conformità per la sicurezza: queste guide all'implementazione di soluzioni illustrano considerazioni relative all'architettura e i passaggi per implementare le funzionalità di sicurezza e conformità.
- Riferimenti sui servizi conformi ai requisiti HIPAA: elenca i servizi HIPAA idonei. Non tutti Servizi AWS sono idonei alla normativa HIPAA.

- AWS Risorse per la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.
- AWS Guide alla conformità dei clienti: comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).
- Valutazione delle risorse con regole nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- AWS Security Hub— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina Documentazione di riferimento sui controlli della Centrale di sicurezza.
- Amazon GuardDuty: Servizio AWS rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, come lo standard PCI DSS, soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- AWS Audit Manager— Ciò Servizio AWS consente di verificare continuamente l' AWS utilizzo per semplificare la gestione del rischio e la conformità alle normative e agli standard di settore.

Resilienza in Amazon Aurora DSQL

L'infrastruttura AWS globale è costruita attorno a zone di disponibilità (Regioni AWS AZ). Regioni AWS forniscono 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, puoi 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 multiregionali offrono tutti i vantaggi della resilienza Multi-AZ, pur garantendo una disponibilità del

database estremamente costante, anche nei casi in cui è inaccessibile ai client delle applicazioni.
Regione AWS

Per ulteriori informazioni sulle zone di disponibilità, Regioni AWS vedere Global Infrastructure.AWS

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 AWSÈ possibile eseguire un backup e un ripristino completi per i cluster a regione singola e multiregione. Per ulteriori informazioni, consulta 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 multiregione offrono funzionalità complete di replica interregionale tra regioni di lettura e scrittura.

Una regione di controllo designata supporta le scritture relative ai soli registri delle transazioni e non consuma spazio di archiviazione. Le regioni di riferimento non dispongono di un endpoint. Ciò significa che le regioni di riferimento 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 ai processori di query 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 ridimensiona automaticamente le repliche di lettura, sia verso l'alto che verso il basso, in base alla frequenza di accesso in lettura.

Le repliche di storage in cluster sono distribuite su un parco di storage multi-tenant. Se un componente o AZ viene danneggiato, 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, Aurora DSQL le aggiunge automaticamente al quorum di storage e le rende disponibili per il cluster.

Elevata disponibilità

Per impostazione predefinita, i cluster a regione singola e multiarea in Aurora DSQL sono attivi e non è necessario effettuare manualmente il provisioning, configurare o riconfigurare alcun cluster. Aurora DSQL automatizza completamente il ripristino dei cluster, eliminando la necessità delle tradizionali operazioni di failover primario-secondario. La replica è sempre sincrona e viene eseguita in modalità multipla AZs, quindi non vi è alcun rischio di perdita dei dati a causa del ritardo della replica o del 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 consente 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 multiregione Aurora DSQL. Ciò significa che i cluster multiregionali 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 il 99,999% per i cluster multiregione.

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. O puoi 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 AWS 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 dell'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 ulteriori informazioni su come connettere Amazon VPC alla rete locale, consulta la Guida per l'utente e la Guida per AWS Direct Connect l'utente AWS Site-to-Site VPN 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 SQL

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 sui cluster SQL `list` Aurora. Consultare 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. Consultare 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 PrivateLink quote 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 disabilitate il DNS privato quando create l'endpoint dell'interfaccia Aurora DSQL. Quando il DNS privato è abilitato, le richieste al servizio Aurora DSQL effettuate dall'interno di 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 DSQL di Aurora effettuate all'interno di 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.

Elenco dei cluster utilizzando un URL di endpoint

Nell'esempio seguente, sostituisci il nome DNS Regione AWS `us-east-1` e il nome DNS dell'`vpce-1a2b3c4d-5e6f.dynamodb.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

Fate riferimento al riferimento sull'API Aurora DSQL per la documentazione sulla gestione delle risorse in Aurora DSQL.

Gestione delle politiche degli endpoint

Testando e configurando a fondo le policy degli endpoint di Amazon VPC, puoi contribuire a garantire che il tuo cluster Aurora DSQL sia sicuro, conforme e allineato ai requisiti di governance e controllo degli accessi specifici della tua organizzazione.

Esempio: policy di accesso SQL Aurora completa

La seguente policy garantisce 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: politica di accesso SQL Aurora con restrizioni

La seguente politica consente solo queste azioni Aurora DSQL.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Tutte le altre azioni 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 PrivateLink endpoint 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

Configurazione di un endpoint di connessione AWS PrivateLink

Fase 1: Ottieni il nome del servizio per il tuo cluster

Quando crei un AWS PrivateLink endpoint per la connessione al cluster, devi prima 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
```

Example response

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

Il nome del servizio include un identificatore, come nell'esempio. `dsql-fnh4` 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: Creare l'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 `--no-private-dns-enabled` flag durante la creazione dell'endpoint, poiché ciò impedirà il corretto funzionamento delle istruzioni di connessione riportate di seguito. Se disabiliti il DNS privato, dovrai creare il tuo record DNS privato 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
```

Example response

```
{
    "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);
```

Connessione a un cluster Aurora DSQL utilizzando un endpoint di connessione AWS PrivateLink

Una volta che l' AWS PrivateLink endpoint 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.

Costruire il 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

Utilizza 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
```

Risoluzione dei problemi relativi a AWS PrivateLink

Problemi e soluzioni comuni

La tabella seguente elenca i problemi e le soluzioni comuni relativi AWS PrivateLink ad Aurora DSQL.

Problema	Causa possibile	Soluzione
Timeout di connessione	Gruppo di sicurezza non configurato correttamente	Usa Amazon VPC Reachability Analyzer per assicurarti che la configurazione della rete consenta il traffico sulla porta 5432.
Errore di risoluzione DNS	DNS privato non abilitato	Verifica che l'endpoint Amazon VPC sia stato creato con DNS privato abilitato.
Errore di autenticazione	Credenziali errate o token scaduto	Genera un nuovo token di autenticazione e verifica il nome utente.
Nome del servizio non trovato	ID cluster errato	Ricontrolla l'ID del cluster e Regione AWS quando recuperi il nome del servizio.

Risorse correlate

Per ulteriori informazioni, consulta le seguenti risorse:

- Guida per l'utente di Amazon Aurora DSQL
- Documentazione AWS PrivateLink
- Accedi ai servizi tramite AWSAWS PrivateLink

Analisi della configurazione e delle vulnerabilità in Amazon Aurora DSQL

AWS gestisce 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 frattempo AWS, 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 `aws:SourceArn`le chiavi di contesto della condizione `aws:SourceAccount`globale nelle politiche delle risorse per limitare le autorizzazioni che Amazon Aurora DSQL fornisce a un altro servizio alla risorsa. Utilizza `aws:SourceArn` se desideri consentire l'associazione di una sola risorsa all'accesso tra servizi. Utilizza `aws:SourceAccount` se desideri 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 usare la chiave di contesto della condizione globale `aws:SourceArn`con l'ARN completo della risorsa. Se non conosci l'ARN completo della risorsa o scegli più risorse, utilizza la chiave di contesto della condizione globale `aws:SourceArn` con caratteri jolly (*) per le parti sconosciute dell'ARN. Ad esempio, `arn:aws:`*`servicename`*`:*:`*`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 come è possibile utilizzare le chiavi di contesto `aws:SourceArn` e `aws:SourceAccount` global condition in Aurora DSQL per evitare il confuso problema del vice.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Best practice di sicurezza per Aurora DSQL

Aurora DSQL offre una serie di funzionalità di sicurezza da considerare durante lo sviluppo e l'implementazione delle proprie politiche 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, gestiscile come considerazioni utili anziché prescrizioni.

Argomenti

Procedure ottimali di sicurezza per Aurora DSQL da parte dei Detective

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.

Etichetta le tue risorse Aurora DSQL per l'identificazione e l'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: utilizzata 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: utilizzata per determinare requisiti come la crittografia.
- Riservatezza: un identificatore per lo specifico livello di riservatezza dei dati supportato da una risorsa.
- Ambiente: utilizzato per distinguere tra infrastruttura di sviluppo, test e produzione.

Per ulteriori informazioni, consulta Best Practices for Tagging AWS Resources.

Best practice di 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.

Usa 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 consente di ottenere chiavi di accesso temporanee che è possibile utilizzare per accedere Servizi AWS alle risorse.

Per ulteriori informazioni, consulta Autenticazione e autorizzazione per Aurora DSQL.

Utilizza le policy IAM per l'autorizzazione di base SQL di Aurora.

Quando concedi le autorizzazioni, decidi chi le ottiene, per quali operazioni dell'API Aurora DSQL ottengono le autorizzazioni e le azioni specifiche che desideri consentire su tali risorse. L'implementazione dei privilegi minimi è fondamentale per ridurre i rischi di sicurezza e l'impatto che può risultare da errori o intenzioni dannose.

Associa le policy di autorizzazione ai ruoli IAM e concedi le autorizzazioni per eseguire operazioni sulle risorse DSQL di Aurora. Sono disponibili anche i limiti di autorizzazione 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 di database personalizzati per gestire e connettersi al cluster. Per ulteriori informazioni, vedere Accesso ad Aurora DSQL e Informazioni sull'autenticazione e l'autorizzazione per Aurora DSQL.

Utilizzo **`verify-full`** in ambienti di produzione.

Questa impostazione verifica che il certificato del server sia firmato da un'autorità di certificazione attendibile e che il nome host del server corrisponda al certificato.

Aggiorna il tuo 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.

Etichettatura delle risorse in Aurora SQL

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 fornite una chiave, il valore è facoltativo.

È possibile utilizzare AWS Management Console AWS CLI, the o the AWS SDKs 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, AWS CLI usa l'`TagResource`operazione.

Etichettare i cluster con un nome

Aurora DSQL crea cluster con un identificatore univoco globale assegnato come Amazon Resource Name (ARN). Se desideri assegnare un nome intuitivo al tuo cluster, ti consigliamo di utilizzare un Tag.

Se si crea una console con la console Aurora DSQL, Aurora DSQL crea automaticamente un tag. Questo tag ha una chiave Name e un valore generato automaticamente che rappresenta il nome del cluster. Questo valore è configurabile, quindi puoi assegnare un nome più intuitivo al tuo cluster. Se un cluster ha 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 chiavi e valori sono lettere, cifre, spazi bianchi e uno qualsiasi dei seguenti simboli: _.:/= + - @.
- Per chiavi e valori viene fatta distinzione tra maiuscole e minuscole.

Note sull'utilizzo dei tag

Quando usi i tag in Aurora DSQL, considera quanto segue.

- Quando utilizzi le operazioni API AWS CLI o Aurora DSQL, assicurati di fornire l'Amazon Resource Name (ARN) per la risorsa Aurora DSQL con cui lavorare. Per ulteriori informazioni, consulta il formato Amazon Resource Name (ARNs) per le risorse Aurora DSQL.
- Ogni risorsa dispone di un set di tag, ovvero una raccolta di uno o più tag assegnati alla risorsa.
- Ogni risorsa può avere fino a 50 tag per set di tag.
- Se elimini una risorsa, i tag associati vengono eliminati.
- Puoi aggiungere tag quando crei una risorsa, puoi visualizzare e modificare i tag utilizzando le seguenti operazioni API: `TagResource`, `UntagResource`, e. `ListTagsForResource`
- Puoi utilizzare i tag con le policy IAM. È possibile utilizzarli per gestire l'accesso ai cluster Aurora DSQL e per controllare quali azioni possono essere applicate a tali risorse. Per ulteriori informazioni, consulta Controllo dell'accesso alle AWS risorse tramite tag.
- Puoi utilizzare i tag per varie altre attività AWS. Per ulteriori informazioni, consulta Strategie di tagging comuni.

Considerazioni sull'utilizzo di Amazon Aurora DSQL

Prendi in considerazione i seguenti comportamenti quando lavori con Amazon Aurora DSQL. Per ulteriori informazioni sulla compatibilità e il supporto di PostgreSQL, consulta. Compatibilità delle funzionalità SQL in Aurora DSQL Per quote e limiti, vedi. Quote di cluster e limiti del database in Amazon Aurora SQL

- Aurora DSQL non completa `COUNT(*)` le operazioni prima del timeout della transazione per tabelle di grandi dimensioni. Per recuperare il conteggio delle righe della tabella dal catalogo di sistema, vedere Utilizzo delle tabelle e dei comandi di sistema in Aurora DSQL.
- Le chiamate dei driver `PG_PREPARED_STATEMENTS` potrebbero fornire una visualizzazione incoerente delle istruzioni preparate memorizzate nella cache per il cluster. Potresti visualizzare 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 collegati in più aree geografiche, 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 riscontri questi errori di transito, riprova la transazione o riconnettiti con il cliente.
- 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 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 SQL

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

Descrizione	Limite predefinito	Configurabile?	Codice di errore Aurora DSQL
Numero massimo di cluster a regione singola per Account AWS	20 cluster	Sì	Codice di errore API `ServiceQuotaExceededExcep`
Numero massimo di cluster multiregionali per Account AWS	5 cluster	Sì	Codice di errore API `ServiceQuotaExceededExcep`
Spazio di archiviazione massimo per cluster	Limite predefinito di 10 TiB, fino a 128 TiB con aumento del limite approvato	Sì	`DISK_FULL(53100)`
Numero massimo di	10.000 connessioni	Sì	`TOO_MANY_CONNECTIONS(53300)`

Descrizione	Limite predefinito	Configurabile?	Codice di errore Aurora DSQL
connessioni per cluster			
Velocità massima di connessione per cluster	100 connessioni al secondo	No	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacità massima di interruzione della connessione 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.

Descrizione	Limite predefinito	Configurabile?	Codice di errore 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

Descrizione	Limite predefinito	Configurabile?	Codice di errore 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. Consultare Considerazioni su Aurora DSQL per la compatibilità con PostgreSQL.	No	54000	ERROR: transaction row limit
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.

Descrizione	Limite predefinito	Configurabile?	Codice di errore Aurora DSQL	Messaggio di errore
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 transazione	5 minuti	No	54000	ERROR: transaction age limit exceeded
Durata massima della connessione	60 minuti	No	Nessun codice di errore	Nessun messaggio di errore
Numero massimo di visualizzazioni in un database	5.000	No	54000	ERROR: creating more than 500 allowed
Dimensione massima della definizione della vista	2 MiB	No	54000	ERROR: view definition too la

Per i limiti dei tipi di dati specifici di Aurora DSQL, vedere. Tipi di dati supportati in Aurora DSQL

Riferimento all'API Aurora DSQL

Oltre a AWS Management Console and the 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, consulta 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, consulta la pagina Errori comuni.

Per ulteriori informazioni su AWS CLI, vedere il AWS Command Line Interface riferimento per Aurora DSQL.

Risoluzione dei problemi in Aurora DSQL

> **Note**
>
> I seguenti argomenti forniscono consigli per la risoluzione di errori e problemi che potrebbero verificarsi durante l'utilizzo di Aurora DSQL. Se trovi un problema che non è elencato qui, contatta l'assistenza AWS

Argomenti

Risoluzione degli errori di connessione

errore: codice di errore SSL non riconosciuto: 6

Causa: potresti utilizzare una versione di psql precedente alla versione 14, che non supporta Server Name Indication (SNI). L'SNI è necessario per la connessione ad Aurora DSQL.

Puoi controllare la versione del tuo client con. `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 degli 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 puoi autenticarti con quel token.

Inoltre, Aurora DSQL rifiuta la richiesta di connessione se il ruolo assunto è scaduto. Ad esempio, se provi a connetterti con un ruolo IAM temporaneo anche se il token di autenticazione non è scaduto, Aurora DSQL rifiuterà la richiesta di connessione.

Per ulteriori informazioni su come IAM funziona con Aurora DSQL, consulta Comprendere l'autenticazione e l'autorizzazione per Aurora DSQL e in Aurora DSQL.AWS Identity and Access Management

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 tua richiesta. Per ulteriori informazioni, consulta Perché le richieste vengono firmate.

Il ruolo IAM non esiste <role>

Aurora DSQL non è riuscita a trovare il tuo ruolo IAM. Per ulteriori informazioni, consulta Ruoli IAM.

Il ruolo IAM deve assomigliare a un ARN IAM

Vedi IAM Identifiers - IAM ARNs per ulteriori informazioni.

Risoluzione degli errori di autorizzazione

Ruolo non supportato <role>

Aurora DSQL non supporta l'operazione. `GRANT` Vedi Sottoinsiemi supportati di comandi PostgreSQL in Aurora DSQL.

Impossibile stabilire un rapporto di fiducia con il ruolo <role>

Aurora DSQL non supporta l'operazione. `GRANT` Vedi Sottoinsiemi supportati di comandi PostgreSQL in Aurora DSQL.

Il ruolo non esiste <role>

Aurora DSQL non è riuscita a trovare l'utente del database specificato. Vedi Autorizzare i ruoli di database personalizzati per la connessione a un cluster.

ERRORE: autorizzazione negata per concedere la fiducia a IAM con ruolo <role>

Per concedere l'accesso a un ruolo del database, devi essere connesso al cluster con il ruolo di amministratore. Per ulteriori informazioni, consulta Autorizzare i ruoli del database a utilizzare SQL in un database.

ERRORE: il ruolo deve avere l'attributo LOGIN <role>

Tutti i ruoli del database che crei devono avere l'`LOGIN`autorizzazione.

Per risolvere questo errore, assicurati di aver creato il ruolo PostgreSQL con l'autorizzazione. `LOGIN` Per ulteriori informazioni, vedere CREATE ROLE e ALTER ROLE nella documentazione di PostgreSQL.

ERRORE: il ruolo non può essere eliminato perché alcuni oggetti dipendono da esso <role>

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 ulteriori 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: SELECT FOR UPDATE in una transazione di sola lettura non è un'operazione

Stai tentando un'operazione che non è consentita in una transazione di sola lettura. Per ulteriori informazioni, consulta Comprendere il controllo della concorrenza in Aurora DSQL.

Errore: usa invece **`CREATE INDEX ASYNC`**

Per creare un indice su una tabella con righe esistenti, è necessario utilizzare il `CREATE INDEX ASYNC` comando. Per ulteriori 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, riprova se necessario»

OC001 «ERRORE: lo schema è stato aggiornato da un'altra transazione, riprova se necessario»

La tua sessione PostgreSQL aveva una copia memorizzata nella cache del catalogo degli schemi. La copia memorizzata nella cache era valida al momento del caricamento. Chiamiamo l'ora T1 e la versione V1.

Un'altra transazione aggiorna il catalogo all'ora T2. Chiamiamola 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 riprovi alla volta T3 dalla sessione originale, Aurora DSQL aggiorna la cache del catalogo. La transazione in T3 utilizza il catalogo V2. Aurora DSQL completerà la transazione a condizione che non siano state apportate altre modifiche al catalogo dal momento del T2.

SSL/TLS Risoluzione dei problemi di connessione

Errore SSL: verifica del certificato non riuscita

Questo errore indica che il client non è in grado di verificare il certificato del server. Assicuratevi che:

1. Il certificato Amazon Root CA 1 è installato correttamente. Consulta Configurazione dei SSL/TLS certificati per le connessioni Aurora DSQL le istruzioni su come convalidare e installare questo certificato.
2. La variabile di `PGSSLROOTCERT` ambiente punta al file di certificato corretto.
3. Il file del certificato dispone delle autorizzazioni corrette.

Codice di errore SSL non riconosciuto: 6

Questo errore si verifica con i client PostgreSQL precedenti alla versione 14. Aggiorna il tuo 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 Connessione da Windows istruzioni.

Cronologia dei documenti per la Guida per l'utente SQL di Amazon Aurora

La tabella seguente descrive le versioni della documentazione per Aurora DSQL.

Modifica	Descrizione	Data
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
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 lavori. 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. AWS Console Home Ciò include l'avvio, l'arresto e il monitoraggio dei lavori. Supporta anche l'utilizzo di chiavi KMS gestite dal cliente per la crittografia e l'avvio dei cluster. AWS CloudShell Per ulteriori informazioni, vedere AmazonAuroraDSQLConsoleFullAccesse Utilizzo 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 AmazonAuroraDSQLReadOnlyAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025

AmazonAuroraDSQLFullAccess – Aggiornamento dell'accesso	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 AmazonAuroraDSQLConsoleFullAccesse Utilizzo 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 multiregione e la connessione endpoint VPC. Le nuove autorizzazioni includono :. `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Vedi AmazonAuroraDSQLConsoleFullAccesse Utilizzo dei ruoli collegati ai servizi in Aurora DSQL.	13 maggio 2025
AuroraDsqlServiceLinkedRolePolicy update	Aggiunge la possibilità di pubblicare metriche nella policy `AWS/AuroraDSQL` e `AWS/Usage` `CloudWatch` namespace nella policy. Ciò consente al servizio o al ruolo associato di emettere 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

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 ulteriori informazioni, consulta Gestione e connessione ai cluster SQL di Amazon Aurora utilizzando. AWS PrivateLink	8 maggio 2025
Versione iniziale	Versione iniziale della Amazon Aurora DSQL User Guide.	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.