aws

Benutzerhandbuch

Amazon Aurora DSQL

Amazon Aurora DSQL: Benutzerhandbuch

Table of Contents

Was ist Amazon Aurora DSQL?

Amazon Aurora DSQL ist ein serverloser, verteilter relationaler Datenbankservice, der für transaktionale Workloads optimiert ist. Aurora DSQL bietet praktisch unbegrenzte Skalierbarkeit und erfordert nicht, dass Sie die Infrastruktur verwalten. Die hochverfügbare Active-Active-Architektur bietet eine Verfügbarkeit von 99,99% in einer Region und 99,999% in mehreren Regionen.

Wann sollte Aurora DSQL verwendet werden

Aurora DSQL ist für transaktionale Workloads optimiert, die von ACID-Transaktionen und einem relationalen Datenmodell profitieren. Da es serverlos ist, eignet sich Aurora DSQL ideal für Anwendungsmuster von Microservice-, serverlosen und ereignisgesteuerten Architekturen. Aurora DSQL ist PostgreSQL-kompatibel, sodass Sie vertraute Treiber, objektrelationale Mappings (), Frameworks und SQL-Funktionen verwenden können. ORMs

Aurora DSQL verwaltet automatisch die Systeminfrastruktur und skaliert Rechenleistung, I/O und Speicher auf der Grundlage Ihrer Arbeitslast. Da Sie keine Server bereitstellen oder verwalten müssen, müssen Sie sich keine Gedanken über Wartungsausfälle im Zusammenhang mit der Bereitstellung, dem Patchen oder Infrastruktur-Upgrades machen.

Aurora DSQL hilft Ihnen bei der Entwicklung und Wartung von Unternehmensanwendungen, die in jeder Größenordnung immer verfügbar sind. Das serverlose Active-Active-Design automatisiert die Wiederherstellung nach einem Ausfall, sodass Sie sich keine Gedanken über herkömmliche Datenbank-Failover machen müssen. Ihre Anwendungen profitieren von Multi-AZ- und Multi-Region-Verfügbarkeit, und Sie müssen sich keine Gedanken über mögliche Konsistenz oder fehlende Daten im Zusammenhang mit Failovers machen.

Die wichtigsten Funktionen von Aurora DSQL

Die folgenden Hauptfunktionen helfen Ihnen bei der Erstellung einer serverlosen verteilten Datenbank zur Unterstützung Ihrer Hochverfügbarkeitsanwendungen:

Verteilte Architektur

Aurora DSQL besteht aus den folgenden mandantenfähigen Komponenten:

- Relay und Konnektivität

- Rechenleistung und Datenbanken
- Transaktionsprotokoll, Parallelitätskontrolle und Isolierung
- Speicher

Eine Steuerungsebene koordiniert die vorhergehenden Komponenten. Jede Komponente bietet Redundanz in drei Availability Zones (AZs) mit automatischer Clusterskalierung und Selbstheilung bei Komponentenausfällen. Weitere Informationen darüber, wie diese Architektur Hochverfügbarkeit unterstützt, finden Sie unter. Resilienz in Amazon Aurora DSQL

Cluster mit einer Region und mehreren Regionen

Aurora DSQL-Cluster bieten die folgenden Vorteile:

- Synchrone Datenreplikation
- Konsistente Lesevorgänge
- Automatische Wiederherstellung nach einem Ausfall
- Datenkonsistenz über mehrere AZs oder Regionen hinweg

Wenn eine Infrastrukturkomponente ausfällt, leitet Aurora DSQL Anfragen ohne manuelles Eingreifen automatisch an eine funktionierende Infrastruktur weiter. Aurora DSQL bietet ACID-Transaktionen (Atomicity, Consistency, Isolation, Durability) mit starker Konsistenz, Snapshot-Isolation, Atomizität sowie AZ- und regionsübergreifender Haltbarkeit.

Peering-Cluster mit mehreren Regionen bieten dieselbe Stabilität und Konnektivität wie Cluster mit nur einer Region. Sie verbessern jedoch die Verfügbarkeit, indem sie zwei regionale Endpunkte anbieten, einen in jeder Peering-Cluster-Region. Beide Endpunkte eines Peering-Clusters stellen eine einzige logische Datenbank dar. Sie sind für gleichzeitige Lese- und Schreibvorgänge verfügbar und bieten eine hohe Datenkonsistenz. Sie können aus Leistungs- und Stabilitätsgründen Anwendungen erstellen, die in mehreren Regionen gleichzeitig ausgeführt werden. Dabei können Sie sicher sein, dass Lesegeräte immer dieselben Daten sehen.

Kompatibilität mit PostgreSQL-Datenbanken

Die verteilte Datenbankschicht (Compute) in Aurora DSQL basiert auf einer aktuellen Hauptversion von PostgreSQL. Sie können mit vertrauten PostgreSQL-Treibern und -Tools eine Verbindung zu Aurora DSQL herstellen, z. `psql` Aurora DSQL ist derzeit mit PostgreSQL Version 16 kompatibel und unterstützt eine Teilmenge von PostgreSQL-Funktionen, Ausdrücken und Datentypen. Weitere Hinweise zu den unterstützten SQL-Funktionen finden Sie unter. Kompatibilität der SQL-Funktionen in Aurora DSQL

Verfügbarkeit in der Region für Aurora DSQL

Mit Amazon Aurora DSQL können Sie Datenbank-Instances für mehrere bereitstellen, um globale Anwendungen AWS-Regionen zu unterstützen und die Anforderungen an die Datenresidenz zu erfüllen. Die regionale Verfügbarkeit bestimmt, wo Sie Aurora DSQL-Datenbankcluster erstellen und verwalten können. Datenbankadministratoren und Anwendungsarchitekten, die hochverfügbare, global verteilte Datenbanksysteme entwerfen müssen, müssen häufig die regionale Unterstützung für ihre Workloads verstehen. Zu den häufigsten Anwendungsfällen gehören die Einrichtung einer regionsübergreifenden Notfallwiederherstellung, die Bereitstellung von Benutzern von geografisch näher gelegenen Datenbankinstanzen zur Verringerung der Latenz und die Aufbewahrung von Datenkopien an bestimmten Standorten aus Compliance-Gründen.

Die folgende Tabelle zeigt, AWS-Regionen wo Aurora DSQL derzeit verfügbar ist, und den jeweiligen AWS-Region Endpunkt.

Unterstützte Geräte AWS-Regionen und Endpunkte

Name der Region	Region	Endpunkt	Protokoll
USA Ost (Nord-Virginia)	us-east-1	dsql.us-east-1.api.aws	HTTPS
USA Ost (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
USA West (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS
Europa (Irland)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europa (London)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europa (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
Asien-Pazifik (Tokio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Asien-Pazifik (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS

Cluster-Verfügbarkeit in mehreren Regionen für Aurora DSQL

Sie können Aurora DSQL-Cluster mit mehreren Regionen innerhalb bestimmter AWS Regionsgruppen erstellen. Jeder Regionensatz gruppiert geografisch verwandte Regionen, die in einem Cluster mit mehreren Regionen zusammenarbeiten können.

US-Regionen

- USA Ost (Nord-Virginia)
- USA Ost (Ohio)
- USA West (Oregon)

Regionen in Asien-Pazifik

- Asien-Pazifik (Osaka)
- Asien-Pazifik (Seoul)
- Asien-Pazifik (Tokio)

Europäische Regionen

- Europa (Irland)
- Europa (London)
- Europa (Paris)

Wichtige Einschränkungen

Cluster mit mehreren Regionen müssen innerhalb einer einzigen Regionengruppe erstellt werden. Sie können beispielsweise keinen Cluster erstellen, der sowohl die Regionen USA Ost (Nord-Virginia) als auch Europa (Irland) umfasst.

> **Important**
>
> Aurora DSQL unterstützt derzeit keine kontinentalübergreifenden Cluster mit mehreren Regionen.

Preise für Aurora DSQL

Kosteninformationen finden Sie unter Aurora DSQL-Preise.

Als nächstes

Informationen zu den Kernkomponenten in Aurora DSQL und zu den ersten Schritten mit dem Service finden Sie im Folgenden:

- Erste Schritte mit Aurora DSQL
- Kompatibilität der SQL-Funktionen in Aurora DSQL
- Zugreifen auf Aurora SQL
- Aurora DSQL und PostgreSQL

Erste Schritte mit Aurora DSQL

Amazon Aurora DSQL ist eine serverlose, verteilte relationale Datenbank, die für transaktionale Workloads optimiert ist. In den folgenden Abschnitten erfahren Sie, wie Sie Aurora DSQL-Cluster mit einer oder mehreren Regionen erstellen, eine Verbindung zu ihnen herstellen und ein Beispielschema erstellen und laden. Sie greifen mit dem auf Cluster zu AWS Management Console und interagieren mit Ihrer Datenbank mithilfe des Dienstprogramms. `psql` Am Ende haben Sie einen funktionierenden Aurora DSQL-Cluster eingerichtet, der für Test- oder Produktionsworkloads einsatzbereit ist.

Themen

Voraussetzungen

Bevor Sie Aurora DSQL verwenden können, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen:

- Ihre IAM-Identität muss über die Berechtigung verfügen, sich bei der anzumelden. AWS Management Console
- Ihre IAM-Identität muss eines der folgenden Kriterien erfüllen:
  - Zugriff zur Ausführung beliebiger Aktionen auf einer beliebigen Ressource in Ihrem AWS-Konto
  - Die IAM-Berechtigung `iam:CreateServiceLinkedRole` und die Möglichkeit, Zugriff auf die IAM-Richtlinienaktion zu erhalten `dsql:*`
- Wenn Sie das AWS CLI in einer UNIX-ähnlichen Umgebung verwenden, stellen Sie sicher, dass Python Version 3.8+ und `psql` Version 14+ installiert sind. Führen Sie die folgenden Befehle aus, um Ihre Anwendungsversionen zu überprüfen.

```
python3 --version
```

```
psql --version
```

Wenn Sie das AWS CLI in einer anderen Umgebung verwenden, stellen Sie sicher, dass Sie Python Version 3.8+ und `psql` Version 14+ manuell einrichten.

- Wenn Sie beabsichtigen, mit Aurora DSQL auf Python zuzugreifen AWS CloudShell, werden Python-Versionen 3.8+ und `psql` Version 14+ ohne zusätzliche Einrichtung bereitgestellt. Weitere Informationen zu finden Sie unter Was AWS CloudShell ist? AWS CloudShell .
- Wenn Sie beabsichtigen, über eine GUI auf Aurora DSQL zuzugreifen, verwenden Sie DBeaver oder JetBrains DataGrip. Weitere Informationen erhalten Sie unter Zugreifen auf Aurora DSQL mit DBeaver und Zugreifen auf Aurora DSQL mit JetBrains DataGrip.

Zugreifen auf Aurora SQL

Sie können mit den folgenden Techniken auf Aurora DSQL zugreifen. Informationen zur Verwendung der CLI APIs SDKs, und finden Sie unterZugreifen auf Aurora SQL.

Themen

- Zugreifen auf Aurora DSQL über AWS Management Console
- Zugreifen auf Aurora DSQL mithilfe von SQL-Clients
- Verwenden des PostgreSQL-Protokolls mit Aurora DSQL

Zugreifen auf Aurora DSQL über AWS Management Console

Sie können auf das AWS Management Console für Aurora DSQL unter https://console.aws.amazon.com/dsql zugreifen.

Zugreifen auf Aurora DSQL mithilfe von SQL-Clients

Aurora DSQL verwendet das PostgreSQL-Protokoll. Verwenden Sie Ihren bevorzugten interaktiven Client, indem Sie ein signiertes IAM-Authentifizierungstoken als Passwort angeben, wenn Sie eine Verbindung zu Ihrem Cluster herstellen. Ein Authentifizierungstoken ist eine eindeutige Zeichenfolge, die Aurora DSQL mithilfe von AWS Signature Version 4 dynamisch generiert.

Aurora DSQL verwendet das Token nur zur Authentifizierung. Das Token hat keinen Einfluss auf die Verbindung, nachdem sie hergestellt wurde. Wenn Sie versuchen, mit einem abgelaufenen

Token erneut eine Verbindung herzustellen, wird die Verbindungsanforderung verweigert. Weitere Informationen finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

Themen

- Zugreifen auf Aurora DSQL mit psql (interaktives PostgreSQL-Terminal)
- Zugreifen auf Aurora DSQL mit DBeaver
- Zugreifen auf Aurora DSQL mit JetBrains DataGrip

Zugreifen auf Aurora DSQL mit psql (interaktives PostgreSQL-Terminal)

Das `psql` Hilfsprogramm ist ein terminalbasiertes Frontend für PostgreSQL. Es ermöglicht Ihnen, Abfragen interaktiv einzugeben, sie an PostgreSQL auszugeben und die Abfrageergebnisse zu sehen. Verwenden Sie den PostgreSQL-Client der Version 17, um die Antwortzeiten für Abfragen zu verbessern.

Laden Sie das Installationsprogramm Ihres Betriebssystems von der PostgreSQL-Downloadseite herunter. Weitere Informationen zu finden Sie `psql` unter https://www.postgresql. org/docs/current/app-psql.htm.

Wenn Sie das bereits AWS CLI installiert haben, verwenden Sie das folgende Beispiel, um eine Verbindung zu Ihrem Cluster herzustellen. Sie können entweder das im Lieferumfang enthaltene `psql` Programm verwenden AWS CloudShell, oder Sie können es `psql` direkt installieren.

```
# 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
```

Zugreifen auf Aurora DSQL mit DBeaver

DBeaver ist ein quelloffenes, GUI-basiertes Datenbanktool. Sie können es verwenden, um eine Verbindung zu Ihrer Datenbank herzustellen und diese zu verwalten. Informationen zum Herunterladen DBeaver finden Sie auf der Download-Seite auf der DBeaver Community-Website. In den folgenden Schritten wird erklärt, wie Sie mithilfe von eine Verbindung zu Ihrem Cluster herstellen DBeaver.

Um eine neue Aurora DSQL-Verbindung einzurichten in DBeaver

1. Wählen Sie Neue Datenbankverbindung.
2. Wählen Sie im Fenster Neue Datenbankverbindung die Option PostgreSQL aus.
3. Wählen Sie auf der Registerkarte Verbindungseinstellungen/Main die Option Connect by: Host und geben Sie die folgenden Informationen ein.

   - Host — Verwenden Sie Ihren Cluster-Endpunkt.

     Datenbank — Geben Sie ein `postgres`

     Authentifizierung — Wählen Sie `Database Native`

     Nutzername — Geben Sie ein `admin`

     Passwort — Generieren Sie ein Authentifizierungstoken. Kopieren Sie das generierte Token und verwenden Sie es als Passwort.
4. Ignorieren Sie alle Warnungen und fügen Sie Ihr Authentifizierungstoken in das Feld DBeaverPasswort ein.

   > **Note**
   >
   > Sie müssen den SSL-Modus in den Client-Verbindungen einstellen. Aurora DSQL unterstütztSSLMODE=`require`. Aurora DSQL erzwingt serverseitig die SSL-Kommunikation und lehnt Verbindungen ohne SSL ab.
5. Sie sollten mit Ihrem Cluster verbunden sein und mit der Ausführung von SQL-Anweisungen beginnen können.

> **Important**
>
> Die DBeaver für die PostgreSQL-Datenbanken bereitgestellten Verwaltungsfunktionen (wie Session Manager und Lock Manager) gelten aufgrund ihrer einzigartigen Architektur nicht für eine Datenbank. Diese Bildschirme sind zwar zugänglich, bieten aber keine zuverlässigen Informationen zum Zustand oder Status der Datenbank.

Ablauf der Authentifizierungsdaten für DBeaver

Etablierte Sitzungen bleiben maximal 1 Stunde lang authentifiziert oder bis die DBeaver Verbindung unterbrochen wird oder ein Timeout auftritt. Um neue Verbindungen herzustellen, geben Sie in den Verbindungseinstellungen im Feld Passwort ein gültiges Authentifizierungstoken ein. Der Versuch, eine neue Sitzung zu öffnen (z. B. um neue Tabellen oder eine neue SQL-Konsole aufzulisten), erzwingt einen neuen Authentifizierungsversuch. Wenn das in den Verbindungseinstellungen konfigurierte Authentifizierungstoken nicht mehr gültig ist, schlägt die neue Sitzung fehl und alle zuvor geöffneten Sitzungen werden DBeaver ungültig. Beachten Sie dies, wenn Sie die Dauer Ihres IAM-Authentifizierungstokens mit der `expires-in` Option wählen.

Zugreifen auf Aurora DSQL mit JetBrains DataGrip

JetBrains DataGrip ist eine plattformübergreifende IDE für die Arbeit mit SQL und Datenbanken, einschließlich PostgreSQL. DataGrip enthält eine robuste GUI mit einem intelligenten SQL-Editor. Gehen Sie zum Herunterladen DataGrip auf die Download-Seite auf der JetBrains Website.

Um eine neue Aurora DSQL-Verbindung einzurichten in JetBrains DataGrip

1. Wählen Sie Neue Datenquelle und dann PostgreSQL.
2. Geben Sie auf der Sources/General Registerkarte Daten die folgenden Informationen ein:
   - Host — Verwenden Sie Ihren Cluster-Endpunkt.

     Port — Aurora DSQL verwendet den PostgreSQL-Standard: `5432`

     Datenbank — Aurora DSQL verwendet den PostgreSQL-Standard von `postgres`

     Authentifizierung — Wählen Sie. `User & Password`

     Nutzername — Geben Sie ein`admin`.

Passwort — Generieren Sie ein Token und fügen Sie es in dieses Feld ein.

URL — Ändern Sie dieses Feld nicht. Es wird basierend auf den anderen Feldern automatisch ausgefüllt.

3. Passwort — Geben Sie dieses an, indem Sie ein Authentifizierungstoken generieren. Kopieren Sie die resultierende Ausgabe des Token-Generators und fügen Sie sie in das Passwortfeld ein.

> **Note**
>
> Sie müssen den SSL-Modus in den Client-Verbindungen einstellen. Aurora DSQL unterstütztPGSSLMODE=require. Aurora DSQL erzwingt serverseitig die SSL-Kommunikation und lehnt Verbindungen ohne SSL ab.

4. Sie sollten mit Ihrem Cluster verbunden sein und mit der Ausführung von SQL-Anweisungen beginnen können:

> **Important**
>
> Einige Ansichten, die DataGrip für die PostgreSQL-Datenbanken bereitgestellt werden (wie Sessions), gelten aufgrund ihrer einzigartigen Architektur nicht für eine Datenbank. Diese Bildschirme sind zwar zugänglich, bieten aber keine zuverlässigen Informationen über die tatsächlichen Sitzungen, die mit der Datenbank verbunden sind.

Ablauf der Authentifizierungsdaten

Etablierte Sitzungen bleiben für maximal 1 Stunde authentifiziert oder bis eine ausdrückliche Trennung oder ein clientseitiges Timeout erfolgt. Wenn neue Verbindungen hergestellt werden müssen, muss ein neues Authentifizierungstoken generiert und im Feld Passwort der Datenquelleneigenschaften bereitgestellt werden. Der Versuch, eine neue Sitzung zu öffnen (z. B. um neue Tabellen oder eine neue SQL-Konsole aufzulisten), erzwingt einen neuen Authentifizierungsversuch. Wenn das in den Verbindungseinstellungen konfigurierte Authentifizierungstoken nicht mehr gültig ist, schlägt die neue Sitzung fehl und alle zuvor geöffneten Sitzungen werden ungültig.

Verwenden des PostgreSQL-Protokolls mit Aurora DSQL

PostgreSQL verwendet ein nachrichtenbasiertes Protokoll für die Kommunikation zwischen Clients und Servern. Das Protokoll wird sowohl über als auch über TCP/IP UNIX-Domain-Sockets unterstützt. Die folgende Tabelle zeigt, wie Aurora DSQL das PostgreSQL-Protokoll unterstützt.

PostgreSQL	Aurora SQL	Hinweise
Rolle (auch bekannt als Benutzer oder Gruppe)	Datenbankrolle	Aurora DSQL erstellt eine Rolle für Sie mit dem Name`admin`. Wenn Sie benutzerdefinierte Datenbankrollen erstellen, müssen Sie diese `admin` Rolle verwenden, um sie IAM-Rollen für die Authentifizierung zuzuordnen, wenn Sie eine Verbindung zu Ihrem Cluster herstellen. Weitere Informationen finden Sie unter Benutzerdefinierte Datenbankrollen konfigurieren.
Host (auch bekannt als Hostname oder Hostspec)	Cluster-Endpunkt	Aurora DSQL Single-Region-Cluster bieten einen einzigen verwalteten Endpunkt und leiten den Datenverkehr automatisch um, wenn innerhalb der Region keine Verfügbarkeit besteht.
Port	N/A — Standard verwenden `5432`	Dies ist der PostgreSQL-Standard.
Datenbank (Datenbankname)	verwenden `postgres`	Aurora DSQL erstellt diese Datenbank für Sie, wenn Sie den Cluster erstellen.
SSL-Modus	SSL ist immer serverseitig aktiviert	In Aurora DSQL unterstützt Aurora DSQL den `require` SSL-Modus. Verbindungen ohne SSL werden von Aurora DSQL abgelehnt.
Passwort	Authentifizierungstoken	Aurora DSQL erfordert temporäre Authentifizierungstoken anstelle von langlebigen Passwörtern. Weitere Informationen hierzu

PostgreSQL	Aurora SQL	Hinweise
		finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

Schritt 1: Erstellen Sie einen Aurora DSQL-Cluster mit einer Region

Die Basiseinheit von Aurora DSQL ist der Cluster, in dem Sie Ihre Daten speichern. In dieser Aufgabe erstellen Sie einen Cluster in einem einzigen AWS-Region.

So erstellen Sie einen Cluster mit einer Region in Aurora DSQL

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Wählen Sie Create Cluster und dann Single-Region aus.
3. (Optional) Wählen Sie in den Clustereinstellungen eine der folgenden Optionen aus:
   - Wählen Sie Verschlüsselungseinstellungen anpassen (erweitert), um eine auszuwählen oder zu erstellen AWS KMS key.
   - Wählen Sie Löschschutz aktivieren aus, um zu verhindern, dass Ihr Cluster durch einen Löschvorgang entfernt wird. Standardmäßig ist der Löschschutz ausgewählt.
4. (Optional) Wählen Sie unter Tags ein Tag für diesen Cluster aus, oder geben Sie es ein.
5. Wählen Sie Cluster erstellen.

Schritt 2: Connect zu Ihrem Aurora DSQL-Cluster her

Ein Cluster-Endpunkt wird automatisch generiert, wenn Sie einen Aurora DSQL-Cluster auf der Grundlage seiner Cluster-ID und Region erstellen. Das Benennungsformat ist`clusterid`.dsql.`region`.on.aws. Ein Client verwendet den Endpunkt, um eine Netzwerkverbindung zu Ihrem Cluster herzustellen.

Die Authentifizierung wird mithilfe von IAM verwaltet, sodass Sie keine Anmeldeinformationen in der Datenbank speichern müssen. Ein Authentifizierungstoken ist eine eindeutige Zeichenfolge, die dynamisch generiert wird. Das Token wird nur zur Authentifizierung verwendet und hat keinen Einfluss auf die Verbindung, nachdem sie hergestellt wurde. Bevor Sie versuchen, eine

Verbindung herzustellen, stellen Sie sicher, dass Ihre IAM-Identität über die entsprechenden `dsql:DbConnectAdmin` Berechtigungen verfügt, wie unter beschriebenVoraussetzungen.

> **Note**
>
> Um die Geschwindigkeit der Datenbankverbindung zu optimieren, verwenden Sie den PostgreSQL-Client der Version 17 und setzen Sie ihn `PGSSLNEGOTIATION` auf direct:. `PGSSLNEGOTIATION=direct`

Um mit einem Authentifizierungstoken eine Verbindung zu Ihrem Cluster herzustellen

1. Wählen Sie in der Aurora DSQL-Konsole den Cluster aus, zu dem Sie eine Verbindung herstellen möchten.
2. Wählen Sie Connect aus.
3. Kopieren Sie den Endpunkt von Endpoint (Host).
4. Vergewissern Sie sich, dass im Abschnitt Authentifizierungstoken (Passwort) die Option Als Administrator Connect ausgewählt ist.
5. Kopieren Sie das generierte Authentifizierungstoken. Dieses Token ist 15 Minuten gültig.
6. Verwenden Sie in der Befehlszeile des Betriebssystems den folgenden Befehl, um Ihren Cluster zu starten `psql` und eine Verbindung zu ihm herzustellen. *`your_cluster_endpoint`*Ersetzen Sie es durch den Cluster-Endpunkt, den Sie zuvor kopiert haben.

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

   Wenn Sie zur Eingabe eines Kennworts aufgefordert werden, geben Sie das Authentifizierungstoken ein, das Sie zuvor kopiert haben. Wenn Sie versuchen, mit einem abgelaufenen Token erneut eine Verbindung herzustellen, wird die Verbindungsanforderung verweigert. Weitere Informationen finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.
7. Drücken Sie die Eingabetaste. Sie sollten eine PostgreSQL-Eingabeaufforderung sehen.

   ```
   postgres=>
   ```

Wenn Sie die Fehlermeldung „Zugriff verweigert“ erhalten, stellen Sie sicher, dass Ihre IAM-Identität über die entsprechende Berechtigung verfügt. `dsql:DbConnectAdmin` Wenn Sie über die entsprechende Berechtigung verfügen und weiterhin die Fehlermeldung „Zugriff verweigert“ erhalten, finden Sie weitere Informationen unter Problembehandlung bei IAM und Wie kann ich Fehler mit einer IAM-Richtlinie beheben, bei der der Zugriff verweigert wurde oder bei nicht autorisierten Vorgängen? .

Schritt 3: Führen Sie SQL-Beispielbefehle in Aurora DSQL aus

Testen Sie Ihren Aurora DSQL-Cluster, indem Sie SQL-Anweisungen ausführen. Die folgenden Beispielanweisungen erfordern die Datendateien mit dem Namen `department-insert-multirow.sql` und`invoice.csv`, die Sie aus dem aurora-dsql-samplesaws-samples/ -Repository herunterladen können. GitHub

So führen Sie SQL-Beispielbefehle in Aurora DSQL aus

1. Erstellen Sie ein Schema mit dem Namen`example`.

```
CREATE SCHEMA example;
```

2. Erstellen Sie eine Rechnungstabelle, die eine automatisch generierte UUID als Primärschlüssel verwendet.

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

3. Erstellen Sie einen sekundären Index, der die leere Tabelle verwendet.

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

4. Erstellen Sie eine Abteilungstabelle.

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

5. Verwenden Sie den Befehl`psql \include`, um die Datei mit dem Namen zu laden`department-insert-multirow.sql`, die Sie aus dem aurora-dsql-samplesaws-

samples/ -Repository heruntergeladen haben. GitHub Ersetzen Sie es *my-path* durch den Pfad zu Ihrer lokalen Kopie.

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

6. Verwenden Sie den Befehl`psql \copy`, um die Datei mit dem Namen zu laden`invoice.csv`, die Sie aus dem aurora-dsql-samplesaws-samples/ -Repository heruntergeladen haben. GitHub Ersetzen Sie es *my-path* durch den Pfad zu Ihrer lokalen Kopie.

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

7. Fragen Sie die Abteilungen ab und sortieren Sie sie nach ihrem Gesamtumsatz.

```
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;
```

Die folgende Beispielausgabe zeigt, dass Abteilung Eins die meisten Verkäufe erzielt.

```
           name            |     sum_amount
---------------------------+--------------------
 Example Department One    |  32628.75608634601
 Example Department Three  |  32427.43955110429
 Example Department Eight  | 32256.810987098102
 Example Department Five   |  31391.14891163639
 Example Department Seven  | 31253.236846746757
 Example Department Six    |  29699.06014910414
 Example Department Two    |  29465.58360076501
 Example Department Four   |  28764.19185819191
(8 rows)
```

Schritt 4: Erstellen Sie einen Cluster mit mehreren Regionen

Wenn Sie einen Cluster mit mehreren Regionen erstellen, geben Sie die folgenden Regionen an:

Entfernte Region

Dies ist die Region, in der Sie einen zweiten Cluster erstellen. Sie erstellen einen zweiten Cluster in dieser Region und verknüpfen ihn mit Ihrem ursprünglichen Cluster. Aurora DSQL repliziert alle Schreibvorgänge auf dem ursprünglichen Cluster auf den Remote-Cluster. Sie können auf jedem Cluster lesen und schreiben.

Region bezeugen

Diese Region empfängt alle Daten, die in den Cluster mit mehreren Regionen geschrieben werden. Zeugenregionen hosten jedoch keine Client-Endpunkte und bieten keinen Zugriff auf Benutzerdaten. In Zeugenregionen wird ein begrenztes Fenster des verschlüsselten Transaktionsprotokolls verwaltet. Dieses Protokoll erleichtert die Wiederherstellung und unterstützt das Transaktions-Quorum, falls eine Region nicht mehr verfügbar ist.

Das folgende Beispiel zeigt, wie Sie einen ersten Cluster und einen zweiten Cluster in einer anderen Region erstellen und dann die beiden Cluster miteinander verbinden, um einen Cluster mit mehreren Regionen zu erstellen. Außerdem werden regionsübergreifende Schreibreplikation und konsistente Lesevorgänge von beiden regionalen Endpunkten demonstriert.

Um einen Cluster mit mehreren Regionen zu erstellen

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Klicken Sie im Navigationsbereich auf Cluster.
3. Wählen Sie Cluster erstellen und dann Multi-Region aus.
4. (Optional) Wählen Sie in den Clustereinstellungen eine der folgenden Optionen für Ihren ersten Cluster aus:
   - Wählen Sie Verschlüsselungseinstellungen anpassen (erweitert), um eine auszuwählen oder zu erstellen AWS KMS key.
   - Wählen Sie Löschschutz aktivieren aus, um zu verhindern, dass Ihr Cluster durch einen Löschvorgang entfernt wird. Standardmäßig ist der Löschschutz ausgewählt.
5. Wählen Sie in den Einstellungen für mehrere Regionen die folgenden Optionen für Ihren ersten Cluster aus:
   - Wählen Sie unter Witness Region eine Region aus. Derzeit werden nur Regionen mit Sitz in den USA für Zeugenregionen in Clustern mit mehreren Regionen unterstützt.

- (Optional) Geben Sie unter Cluster-ARN für Remote-Region einen ARN für einen vorhandenen Cluster in einer anderen Region ein. Wenn kein Cluster vorhanden ist, der als zweiter Cluster in Ihrem Cluster mit mehreren Regionen dienen könnte, schließen Sie die Einrichtung ab, nachdem Sie den ersten Cluster erstellt haben.

6. (Optional) Wählen Sie Tags für Ihren ersten Cluster aus.
7. Wählen Sie Cluster erstellen aus, um Ihren ersten Cluster zu erstellen. Wenn Sie im vorherigen Schritt keinen ARN eingegeben haben, zeigt die Konsole die Benachrichtigung Cluster-Setup ausstehend an.
8. Wählen Sie in der Benachrichtigung „Cluster-Setup ausstehend“ die Option Complete Multi-Region-Cluster-Setup aus. Diese Aktion initiiert die Erstellung eines zweiten Clusters in einer anderen Region.
9. Wählen Sie eine der folgenden Optionen für Ihren zweiten Cluster:

   - Cluster-ARN für Remote-Region hinzufügen — Wählen Sie diese Option, wenn ein Cluster vorhanden ist und Sie möchten, dass es der zweite Cluster in Ihrem Cluster mit mehreren Regionen ist.
   - Cluster in einer anderen Region erstellen — Wählen Sie diese Option, um einen zweiten Cluster zu erstellen. Wählen Sie unter Remote-Region die Region für diesen zweiten Cluster aus.

10. Wählen Sie Cluster erstellen in ***your-second-region***, wo *your-second-region* sich der Standort Ihres zweiten Clusters befindet. Die Konsole wird in Ihrer zweiten Region geöffnet.
11. (Optional) Wählen Sie die Clustereinstellungen für Ihren zweiten Cluster aus. Sie können beispielsweise eine auswählen AWS KMS key.
12. Wählen Sie Cluster erstellen, um Ihren zweiten Cluster zu erstellen.
13. Wählen Sie Peer in ***initial-cluster-region***. Dabei *initial-cluster-region* handelt es sich um die Region, die den ersten Cluster hostet, den Sie erstellt haben.
14. Wenn Sie dazu aufgefordert werden, wählen Sie Bestätigen. Mit diesem Schritt ist die Erstellung Ihres Clusters mit mehreren Regionen abgeschlossen.

Um eine Verbindung zu Ihrem zweiten Cluster herzustellen

1. Öffnen Sie die Aurora DSQL-Konsole und wählen Sie die Region für Ihren zweiten Cluster aus.
2. Wählen Sie Clusters (Cluster) aus.
3. Wählen Sie die Zeile für den zweiten Cluster in Ihrem Multi-Region-Cluster aus.

4. Wählen Sie unter Aktionen die Option Öffnen in CloudShell aus.
5. Wählen Sie Als Administrator Connect.
6. Wählen Sie Launch CloudShell (Starten) aus.
7. Wählen Sie Ausführen aus.
8. Erstellen Sie ein Beispielschema, indem Sie die Schritte unter befolgenSchritt 3: Führen Sie SQL-Beispielbefehle in Aurora DSQL aus.

   Beispieltransaktionen

   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. Verwenden Sie die `include` Befehle `psql` `copy` und zum Laden von Beispieldaten. Weitere Informationen finden Sie unter Schritt 3: Führen Sie SQL-Beispielbefehle in Aurora DSQL aus.

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

Um Daten im zweiten Cluster aus der Region abzufragen, in der sich Ihr erster Cluster befindet

1. Wählen Sie in der Aurora DSQL-Konsole die Region für Ihren ersten Cluster aus.
2. Wählen Sie Clusters (Cluster) aus.
3. Wählen Sie die Zeile für den zweiten Cluster in Ihrem Cluster mit mehreren Regionen aus.
4. Wählen Sie unter Aktionen die Option Öffnen in CloudShell aus.
5. Wählen Sie Als Administrator Connect.
6. Wählen Sie Launch CloudShell (Starten) aus.
7. Wählen Sie Ausführen aus.
8. Fragen Sie die Daten ab, die Sie in den zweiten Cluster eingefügt haben.

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;
```

Authentifizierung und Autorisierung für Aurora DSQL

Aurora DSQL verwendet IAM-Rollen und -Richtlinien für die Cluster-Autorisierung. Sie ordnen IAM-Rollen PostgreSQL-Datenbankrollen für die Datenbankautorisierung zu. Dieser Ansatz kombiniert die Vorteile von IAM mit PostgreSQL-Rechten. Aurora DSQL verwendet diese Funktionen, um eine umfassende Autorisierungs- und Zugriffsrichtlinie für Ihren Cluster, Ihre Datenbank und Ihre Daten bereitzustellen.

Verwaltung Ihres Clusters mithilfe von IAM

Verwenden Sie IAM für die Authentifizierung und Autorisierung, um Ihren Cluster zu verwalten:

IAM-Authentifizierung

Um Ihre IAM-Identität bei der Verwaltung von Aurora DSQL-Clustern zu authentifizieren, müssen Sie IAM verwenden. Sie können die Authentifizierung mit dem AWS Management Console, oder dem SDK AWS CLIbereitstellen.AWS

IAM-Autorisierung

Um Aurora DSQL-Cluster zu verwalten, gewähren Sie die Autorisierung mithilfe von IAM-Aktionen für Aurora DSQL. Um beispielsweise einen Cluster zu beschreiben, stellen Sie sicher, dass Ihre IAM-Identität über Berechtigungen für die IAM-Aktion verfügt`dsql:GetCluster`, wie in der folgenden Beispiel-Richtlinienaktion dargestellt.

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

Weitere Informationen finden Sie unter Verwenden von IAM-Richtlinienaktionen zur Verwaltung von Clustern.

Mithilfe von IAM eine Verbindung zu Ihrem Cluster herstellen

Um eine Verbindung zu Ihrem Cluster herzustellen, verwenden Sie IAM für die Authentifizierung und Autorisierung:

IAM-Authentifizierung

Generieren Sie ein temporäres Authentifizierungstoken mithilfe einer IAM-Identität mit Autorisierung, eine Verbindung zu Ihrem Cluster herzustellen. Weitere Informationen hierzu finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

IAM-Autorisierung

Erteilen Sie der IAM-Identität, mit der Sie die Verbindung zum Endpunkt Ihres Clusters herstellen, die folgenden IAM-Richtlinienaktionen:

- Verwenden Sie`dsql:DbConnectAdmin`, wenn Sie die `admin` Rolle verwenden. Aurora DSQL erstellt und verwaltet diese Rolle für Sie. Das folgende Beispiel für eine IAM-Richtlinienaktion ermöglicht das Herstellen `admin` einer Verbindung zu. *`my-cluster`*

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

- Verwenden Sie `dsql:DbConnect` diese Option, wenn Sie eine benutzerdefinierte Datenbankrolle verwenden. Sie erstellen und verwalten diese Rolle mithilfe von SQL-Befehlen in Ihrer Datenbank. Mit der folgenden IAM-Richtlinienaktion kann eine benutzerdefinierte Datenbankrolle bis zu einer Stunde lang eine Verbindung herstellen. *`my-cluster`*

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

Nachdem Sie eine Verbindung hergestellt haben, wird Ihre Rolle für eine Dauer von bis zu einer Stunde für die Verbindung autorisiert.

Interaktion mit Ihrer Datenbank mithilfe von PostgreSQL-Datenbankrollen und IAM-Rollen

PostgreSQL verwaltet Datenbankzugriffsberechtigungen mithilfe des Rollenkonzepts. Eine Rolle kann entweder als Datenbankbenutzer oder als Gruppe von Datenbankbenutzern betrachtet werden,

je nachdem, wie die Rolle eingerichtet ist. Sie erstellen PostgreSQL-Rollen mithilfe von SQL-Befehlen. Um die Autorisierung auf Datenbankebene zu verwalten, gewähren Sie Ihren PostgreSQL-Datenbankrollen PostgreSQL-Berechtigungen.

Aurora DSQL unterstützt zwei Arten von Datenbankrollen: `admin` Rollen und benutzerdefinierte Rollen. Aurora DSQL erstellt automatisch eine vordefinierte `admin` Rolle für Sie in Ihrem Aurora DSQL-Cluster. Sie können die Rolle nicht ändern. `admin` Wenn Sie eine Verbindung zu Ihrer Datenbank herstellen als`admin`, können Sie SQL ausgeben, um neue Rollen auf Datenbankebene zu erstellen, die Sie Ihren IAM-Rollen zuordnen können. Damit IAM-Rollen eine Verbindung zu Ihrer Datenbank herstellen können, ordnen Sie Ihre benutzerdefinierten Datenbankrollen Ihren IAM-Rollen zu.

Authentifizierung

Verwenden Sie die `admin` Rolle, um eine Verbindung zu Ihrem Cluster herzustellen. Nachdem Sie Ihre Datenbank verbunden haben, verwenden Sie den Befehl, `AWS IAM GRANT` um der IAM-Identität, die autorisiert ist, eine Verbindung mit dem Cluster herzustellen, eine benutzerdefinierte Datenbankrolle zuzuordnen, wie im folgenden Beispiel.

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

Weitere Informationen hierzu finden Sie unter Autorisieren von Datenbankrollen, um eine Verbindung zu Ihrem Cluster herzustellen.

Autorisierung

Verwenden Sie die `admin` Rolle, um eine Verbindung zu Ihrem Cluster herzustellen. Führen Sie SQL-Befehle aus, um benutzerdefinierte Datenbankrollen einzurichten und Berechtigungen zu gewähren. Weitere Informationen finden Sie unter PostgreSQL-Datenbankrollen und PostgreSQL-Rechte in der PostgreSQL-Dokumentation.

Verwenden von IAM-Richtlinienaktionen mit Aurora DSQL

Welche IAM-Richtlinienaktion Sie verwenden, hängt von der Rolle ab, die Sie für die Verbindung mit Ihrem Cluster verwenden: entweder `admin` oder eine benutzerdefinierte Datenbankrolle. Die Richtlinie hängt auch von den IAM-Aktionen ab, die für diese Rolle erforderlich sind.

Verwenden von IAM-Richtlinienaktionen zum Herstellen einer Verbindung zu Clustern

Wenn Sie mit der Standard-Datenbankrolle von eine Verbindung zu Ihrem Cluster herstellen`admin`, verwenden Sie eine IAM-Identität mit Autorisierung, um die folgende IAM-Richtlinienaktion auszuführen.

```
"dsql:DbConnectAdmin"
```

Wenn Sie mit einer benutzerdefinierten Datenbankrolle eine Verbindung zu Ihrem Cluster herstellen, ordnen Sie zuerst die IAM-Rolle der Datenbankrolle zu. Die IAM-Identität, die Sie für die Verbindung mit Ihrem Cluster verwenden, muss autorisiert sein, um die folgende IAM-Richtlinienaktion auszuführen.

```
"dsql:DbConnect"
```

Weitere Informationen zu benutzerdefinierten Datenbankrollen finden Sie unter. Verwenden von Datenbankrollen und IAM-Authentifizierung

Verwenden von IAM-Richtlinienaktionen zur Verwaltung von Clustern

Geben Sie bei der Verwaltung Ihrer Aurora DSQL-Cluster Richtlinienaktionen nur für die Aktionen an, die Ihre Rolle ausführen muss. Wenn Ihre Rolle beispielsweise nur Clusterinformationen abrufen muss, können Sie die Rollenberechtigungen auf die `ListClusters` Berechtigungen `GetCluster` und beschränken, wie in der folgenden Beispielrichtlinie

JSON

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

```
    }
  ]
}
```

Die folgende Beispielrichtlinie zeigt alle verfügbaren IAM-Richtlinienaktionen für die Verwaltung von Clustern.

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" : "*"
    }
  ]
}
```

Widerrufen der Autorisierung mit IAM und PostgreSQL

Sie können Ihren IAM-Rollen die Zugriffsberechtigungen für Ihre Rollen auf Datenbankebene entziehen:

Widerrufen der Administratorautorisierung für die Verbindung zu Clustern

Um die Autorisierung für die Verbindung mit Ihrem Cluster mit der `admin` Rolle zu widerrufen, widerrufen Sie den Zugriff der IAM-Identität auf. `dsql:DbConnectAdmin` Bearbeiten Sie entweder die IAM-Richtlinie oder trennen Sie die Richtlinie von der Identität.

Nach dem Widerruf der Verbindungsautorisierung für die IAM-Identität lehnt Aurora DSQL alle neuen Verbindungsversuche von dieser IAM-Identität ab. Alle aktiven Verbindungen, die die IAM-Identität verwenden, bleiben möglicherweise für die Dauer der Verbindung autorisiert. Weitere Informationen zur Verbindungsdauer finden Sie unter Kontingente und Beschränkungen.

Widerrufen der Autorisierung für benutzerdefinierte Rollen zum Herstellen einer Verbindung zu Clustern

Um den Zugriff auf andere Datenbankrollen als zu widerrufen`admin`, widerrufen Sie den Zugriff der IAM-Identität auf. `dsql:DbConnect` Bearbeiten Sie entweder die IAM-Richtlinie oder trennen Sie die Richtlinie von der Identität.

Sie können die Zuordnung zwischen der Datenbankrolle und IAM auch entfernen, indem Sie den Befehl `AWS IAM REVOKE` in Ihrer Datenbank verwenden. Weitere Informationen zum Widerrufen des Zugriffs auf Datenbankrollen finden Sie unter. Widerrufen der Datenbankautorisierung für eine IAM-Rolle

Sie können die Berechtigungen der vordefinierten `admin` Datenbankrolle nicht verwalten. Informationen zum Verwalten von Berechtigungen für benutzerdefinierte Datenbankrollen finden Sie unter PostgreSQL-Rechte. Änderungen an den Rechten werden bei der nächsten Transaktion wirksam, nachdem Aurora DSQL die Änderungstransaktion erfolgreich festgeschrieben hat.

Generieren eines Authentifizierungstokens in Amazon Aurora DSQL

Um mit einem SQL-Client eine Verbindung zu Amazon Aurora DSQL herzustellen, generieren Sie ein Authentifizierungstoken, das als Passwort verwendet wird. Dieses Token wird nur zur Authentifizierung der Verbindung verwendet. Nachdem die Verbindung hergestellt wurde, bleibt die Verbindung gültig, auch wenn das Authentifizierungstoken abläuft.

Wenn Sie mit der AWS Konsole ein Authentifizierungstoken erstellen, läuft das Token standardmäßig automatisch in einer Stunde ab. Wenn Sie das AWS CLI oder verwenden SDKs , um das Token zu erstellen, beträgt die Standardeinstellung 15 Minuten. Die maximale Dauer beträgt 604.800 Sekunden, was einer Woche entspricht. Um von Ihrem Client aus erneut eine Verbindung zu Aurora DSQL herzustellen, können Sie dasselbe Authentifizierungstoken verwenden, falls es nicht abgelaufen ist, oder Sie können ein neues generieren.

Um mit der Generierung eines Tokens zu beginnen, erstellen Sie eine IAM-Richtlinie und einen Cluster in Aurora DSQL. Verwenden Sie dann die AWS Konsole, oder die AWS CLI, um ein AWS SDKs Token zu generieren.

Je nachdem, welche Datenbankrolle Sie für die Verbindung verwendenMithilfe von IAM eine Verbindung zu Ihrem Cluster herstellen, benötigen Sie mindestens die unter aufgeführten IAM-Berechtigungen.

Themen

- Verwenden Sie die AWS Konsole, um ein Authentifizierungstoken in Aurora DSQL zu generieren
- Wird verwendet AWS CloudShell , um ein Authentifizierungstoken in Aurora DSQL zu generieren
- Verwenden Sie das AWS CLI , um ein Authentifizierungstoken in Aurora DSQL zu generieren
- Verwenden Sie das SDKs , um ein Token in Aurora DSQL zu generieren

Verwenden Sie die AWS Konsole, um ein Authentifizierungstoken in Aurora DSQL zu generieren

Aurora DSQL authentifiziert Benutzer mit einem Token und nicht mit einem Passwort. Sie können das Token von der Konsole aus generieren.

Um ein Authentifizierungstoken zu generieren

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Wählen Sie die Cluster-ID des Clusters aus, für den Sie ein Authentifizierungstoken generieren möchten. Wenn Sie noch keinen Cluster erstellt haben, folgen Sie den Schritten unter Schritt 1: Erstellen Sie einen Aurora DSQL-Cluster mit einer Region oderSchritt 4: Erstellen Sie einen Cluster mit mehreren Regionen.
3. Wählen Sie Connect und dann Get Token aus.
4. Wählen Sie aus, ob Sie eine Verbindung als `admin` oder mit einer benutzerdefinierten Datenbankrolle herstellen möchten.
5. Kopieren Sie das generierte Authentifizierungstoken und verwenden Sie es fürZugreifen auf Aurora DSQL mithilfe von SQL-Clients.

Weitere Informationen zu benutzerdefinierten Datenbankrollen und IAM in Aurora DSQL finden Sie unter. Authentifizierung und Autorisierung

Wird verwendet AWS CloudShell , um ein Authentifizierungstoken in Aurora DSQL zu generieren

Bevor Sie mit ein Authentifizierungstoken generieren können AWS CloudShell, stellen Sie sicher, dass Sie einen Aurora DSQL-Cluster erstellen.

Um ein Authentifizierungstoken zu generieren, verwenden Sie AWS CloudShell

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Wählen AWS CloudShell Sie unten links in der AWS Konsole.
3. Führen Sie den folgenden Befehl aus, um ein Authentifizierungstoken für die `admin` Rolle zu generieren. *`us-east-1`*Ersetzen Sie es durch Ihre Region und *`your_cluster_endpoint`* durch den Endpunkt Ihres eigenen Clusters.

   > **Note**
   >
   > Wenn Sie keine Verbindung herstellen als`admin`, verwenden Sie `generate-db-connect-auth-token` stattdessen.

   ```
   aws dsql generate-db-connect-admin-auth-token \
     --expires-in 3600 \
     --region us-east-1 \
     --hostname your_cluster_endpoint
   ```

   Wenn Sie auf Probleme stoßen, finden Sie weitere Informationen unter Problembehandlung bei IAM und Wie kann ich Fehler mit einer IAM-Richtlinie beheben, bei denen der Zugriff verweigert wurde oder bei nicht autorisierten Vorgängen? .
4. Verwenden Sie den folgenden Befehl, `psql` um eine Verbindung zu Ihrem Cluster herzustellen.

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

```
  --host cluster_endpoint
```

5. Sie sollten aufgefordert werden, ein Passwort einzugeben. Kopieren Sie das Token, das Sie generiert haben, und stellen Sie sicher, dass Sie keine zusätzlichen Leerzeichen oder Zeichen verwenden. Fügen Sie es in die folgende Eingabeaufforderung von ein`psql`.

```
Password for user admin:
```

6. Drücken Sie die Eingabetaste. Sie sollten eine PostgreSQL-Eingabeaufforderung sehen.

```
postgres=>
```

   Wenn Sie die Fehlermeldung „Zugriff verweigert“ erhalten, stellen Sie sicher, dass Ihre IAM-Identität über die entsprechende Berechtigung verfügt. `dsql:DbConnectAdmin` Wenn Sie über die entsprechende Berechtigung verfügen und weiterhin die Fehlermeldung „Zugriff verweigert“ erhalten, finden Sie weitere Informationen unter Problembehandlung bei IAM und Wie kann ich Fehler mit einer IAM-Richtlinie beheben, bei der der Zugriff verweigert wurde oder bei nicht autorisierten Vorgängen? .

Weitere Informationen zu benutzerdefinierten Datenbankrollen und IAM in Aurora DSQL finden Sie unter. Authentifizierung und Autorisierung

Verwenden Sie das AWS CLI , um ein Authentifizierungstoken in Aurora DSQL zu generieren

Wenn Ihr Cluster vorhanden ist`ACTIVE`, können Sie mit dem `aws dsql` Befehl ein Authentifizierungstoken auf der CLI generieren. Verwenden Sie eine der folgenden Techniken:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie die `generate-db-connect-admin-auth-token` Option.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie die `generate-db-connect-auth-token` Option.

Im folgenden Beispiel werden die folgenden Attribute verwendet, um ein Authentifizierungstoken für die `admin` Rolle zu generieren.

- *your_cluster_endpoint*— Der Endpunkt des Clusters. Es folgt dem Format*your_cluster_identifier*.dsql.*region*.on.aws, wie im Beispiel01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws.
- *region*— Die AWS-Region, wie us-east-2 oderus-east-1.

In den folgenden Beispielen wird festgelegt, dass die Ablaufzeit für das Token in 3600 Sekunden (1 Stunde) abläuft.

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
```

Verwenden Sie das SDKs , um ein Token in Aurora DSQL zu generieren

Sie können ein Authentifizierungstoken für Ihren Cluster generieren, wenn er sich im ACTIVE Status befindet. Die SDK-Beispiele verwenden die folgenden Attribute, um ein Authentifizierungstoken für die admin Rolle zu generieren:

- *your_cluster_endpoint*(oder*yourClusterEndpoint*) — Der Endpunkt Ihres Aurora DSQL-Clusters. Das Benennungsformat ist *your_cluster_identifier*.dsql.*region*.on.aws wie im Beispiel01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws.
- *region*(oder*RegionEndpoint*) — Das, AWS-Region in dem sich Ihr Cluster befindet, z. B. us-east-2 oderus-east-1.

Python SDK

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`generate_db_connect_admin_auth_token`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`GenerateDBConnectAdminAuthToken`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`getDbConnectAdminAuthToken`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`generateDbConnectAdminAuthToken`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`db_connect_admin_auth_token`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`generate_db_connect_admin_auth_token`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`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**
>
> Das offizielle SDK for .NET enthält keinen integrierten API-Aufruf zur Generierung eines Authentifizierungstokens für Aurora DSQL. Stattdessen müssen Sie, eine Utility-Klasse`DSQLAuthTokenGenerator`, verwenden. Das folgende Codebeispiel zeigt, wie das Authentifizierungstoken für.NET generiert wird.

Sie können das Token auf folgende Weise generieren:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie`DbConnectAdmin`.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie`DbConnect`.

Im folgenden Beispiel wird die `DSQLAuthTokenGenerator` Utility-Klasse verwendet, um das Authentifizierungstoken für einen Benutzer mit der `admin` Rolle zu generieren. Ersetzen Sie es *`insert-dsql-cluster-endpoint`* durch Ihren Cluster-Endpunkt.

```
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**
>
> Das Golang SDK bietet keine integrierte Methode zum Generieren eines vorsignierten Tokens. Sie müssen die signierte Anfrage manuell erstellen, wie im folgenden Codebeispiel gezeigt.

Geben Sie im folgenden Codebeispiel den auf dem PostgreSQL `action` basierenden Benutzer an:

- Wenn Sie eine Verbindung mit der `admin` Rolle herstellen, verwenden Sie die `DbConnectAdmin` Aktion.
- Wenn Sie eine Verbindung mit einer benutzerdefinierten Datenbankrolle herstellen, verwenden Sie die `DbConnect` Aktion.

Zusätzlich zu *`yourClusterEndpoint`* und *`region`* verwendet das folgende Beispiel*`action`*. Geben Sie den auf dem PostgreSQL *`action`* basierenden Benutzer an.

```
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
}
```

Verwenden von Datenbankrollen und IAM-Authentifizierung

Aurora DSQL unterstützt die Authentifizierung sowohl mit IAM-Rollen als auch mit IAM-Benutzern. Sie können beide Methoden verwenden, um Aurora DSQL-Datenbanken zu authentifizieren und darauf zuzugreifen.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über bestimmte Berechtigungen verfügt, aber keiner bestimmten Person zugeordnet ist. Mithilfe von IAM-Rollen werden temporäre Sicherheitsanmeldedaten bereitgestellt. Sie können vorübergehend eine IAM-Rolle auf verschiedene Arten übernehmen:

- Durch den Rollenwechsel in der AWS Management Console
- Durch Aufrufen einer AWS CLI oder AWS API-Operation
- Durch die Verwendung einer benutzerdefinierten URL

Nachdem Sie eine Rolle übernommen haben, können Sie mit den temporären Anmeldeinformationen der Rolle auf Aurora DSQL zugreifen. Weitere Informationen zu Methoden zur Verwendung von Rollen finden Sie unter IAM-Identitäten im IAM-Benutzerhandbuch.

IAM-Benutzer

Ein IAM-Benutzer ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über bestimmte Berechtigungen verfügt und einer einzelnen Person oder Anwendung zugeordnet ist. IAM-Benutzer verfügen über langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel, die für den Zugriff auf Aurora DSQL verwendet werden können.

> **Note**
>
> Um SQL-Befehle mit IAM-Authentifizierung auszuführen, können Sie in den folgenden Beispielen entweder die IAM-Rolle ARNs oder den IAM-Benutzer ARNs verwenden.

Autorisieren von Datenbankrollen, um eine Verbindung zu Ihrem Cluster herzustellen

Erstellen Sie eine IAM-Rolle und gewähren Sie die Verbindungsautorisierung mit der IAM-Richtlinienaktion:. `dsql:DbConnect`

Die IAM-Richtlinie muss auch die Erlaubnis für den Zugriff auf die Clusterressourcen gewähren. Verwenden Sie einen Platzhalter (*) oder folgen Sie den Anweisungen unter Verwenden von IAM-Bedingungsschlüsseln mit Amazon Aurora DSQL.

Autorisieren von Datenbankrollen zur Verwendung von SQL in Ihrer Datenbank

Sie müssen eine IAM-Rolle mit Autorisierung verwenden, um eine Verbindung zu Ihrem Cluster herzustellen.

1. Stellen Sie mithilfe eines SQL-Dienstprogramms eine Connect zu Ihrem Aurora DSQL-Cluster her.

   Verwenden Sie die `admin` Datenbankrolle mit einer IAM-Identität, die für IAM-Aktionen autorisiert ist, um eine Verbindung `dsql:DbConnectAdmin` zu Ihrem Cluster herzustellen.

2. Erstellen Sie eine neue Datenbankrolle und achten Sie darauf, die `WITH LOGIN` Option anzugeben.

   ```
   CREATE ROLE example WITH LOGIN;
   ```

3. Ordnen Sie die Datenbankrolle der IAM-Rolle ARN zu.

   ```
   AWS IAM GRANT example TO 'arn:aws:iam::012345678912:role/example';
   ```

4. Erteilen Sie der Datenbankrolle Berechtigungen auf Datenbankebene

   In den folgenden Beispielen wird der GRANT Befehl verwendet, um die Autorisierung innerhalb der Datenbank bereitzustellen.

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

Weitere Informationen finden Sie unter PostgreSQL GRANT und PostgreSQL-Privilegien in der PostgreSQL-Dokumentation.

Zuordnungen von IAM zu Datenbankrollen anzeigen

Um die Zuordnungen zwischen IAM-Rollen und Datenbankrollen anzuzeigen, fragen Sie die Systemtabelle ab. `sys.iam_pg_role_mappings`

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

Beispielausgabe:

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

Diese Tabelle zeigt alle Zuordnungen zwischen IAM-Rollen (identifiziert durch ihren ARN) und PostgreSQL-Datenbankrollen.

Widerrufen der Datenbankautorisierung für eine IAM-Rolle

Verwenden Sie den `AWS IAM REVOKE` Vorgang, um die Datenbankautorisierung zu widerrufen.

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

Weitere Informationen zum Widerrufen der Autorisierung finden Sie unterWiderrufen der Autorisierung mit IAM und PostgreSQL.

Aurora DSQL und PostgreSQL

Aurora DSQL ist eine PostgreSQL-kompatible, verteilte relationale Datenbank, die für transaktionale Workloads entwickelt wurde. Aurora DSQL verwendet zentrale PostgreSQL-Komponenten wie den Parser, den Planer, den Optimierer und das Typsystem.

Das Aurora DSQL-Design stellt sicher, dass die gesamte unterstützte PostgreSQL-Syntax kompatibles Verhalten bietet und identische Abfrageergebnisse liefert. Aurora DSQL bietet beispielsweise Typkonvertierungen, arithmetische Operationen sowie numerische Präzision und Skalierung, die mit PostgreSQL identisch sind. Alle Abweichungen werden dokumentiert.

Aurora DSQL bietet auch erweiterte Funktionen wie optimistische Parallelitätssteuerung und verteiltes Schema-Management. Mit diesen Funktionen können Sie die vertrauten Tools von PostgreSQL verwenden und gleichzeitig von der Leistung und Skalierbarkeit profitieren, die für moderne, cloudnative, verteilte Anwendungen erforderlich sind.

Höhepunkte der PostgreSQL-Kompatibilität

Aurora DSQL basiert derzeit auf PostgreSQL Version 16. Zu den wichtigsten Kompatibilitäten gehören die folgenden:

Wire-Protokoll

Aurora DSQL verwendet das standardmäßige PostgreSQL v3-Wire-Protokoll. Dies ermöglicht die Integration mit standardmäßigen PostgreSQL-Clients, -Treibern und -Tools. Aurora DSQL ist beispielsweise kompatibel mit `psql`, `pgjdbc`, und`psycopg`.

SQL-Kompatibilität

Aurora DSQL unterstützt eine Vielzahl von Standard-PostgreSQL-Ausdrücken und -Funktionen, die häufig in transaktionalen Workloads verwendet werden. Unterstützte SQL-Ausdrücke liefern identische Ergebnisse wie PostgreSQL, einschließlich der folgenden:

- Umgang mit Nullen
- Verhalten bei der Sortierreihenfolge
- Skalierung und Präzision für numerische Operationen
- Äquivalenz für Zeichenkettenoperationen

Weitere Informationen finden Sie unter Kompatibilität der SQL-Funktionen in Aurora DSQL.

Verwaltung von Transaktionen

Aurora DSQL behält die Hauptmerkmale von PostgreSQL bei, wie z. B. ACID-Transaktionen und eine Isolationsstufe, die PostgreSQL Repeatable Read entspricht. Weitere Informationen finden Sie unter Parallelitätssteuerung in Aurora DSQL.

Die wichtigsten architektonischen Unterschiede

Das verteilte Shared-Nothing-Design von Aurora DSQL führt zu einigen grundlegenden Unterschieden gegenüber herkömmlichem PostgreSQL. Diese Unterschiede sind integraler Bestandteil der Aurora DSQL-Architektur und bieten viele Leistungs- und Skalierbarkeitsvorteile. Zu den wichtigsten Unterschieden gehören die folgenden:

Optimistische Parallelitätskontrolle (OCC)

Aurora DSQL verwendet ein optimistisches Parallelitätskontrollmodell. Dieser Ansatz ohne Sperren verhindert, dass sich Transaktionen gegenseitig blockieren, beseitigt Deadlocks und ermöglicht eine parallel Ausführung mit hohem Durchsatz. Diese Funktionen machen Aurora DSQL besonders wertvoll für Anwendungen, die eine konsistente Leistung in großem Maßstab erfordern. Weitere Beispiele finden Sie unterParallelitätssteuerung in Aurora DSQL.

Asynchrone DDL-Operationen

Aurora DSQL führt DDL-Operationen asynchron aus, was unterbrechungsfreie Lese- und Schreibvorgänge bei Schemaänderungen ermöglicht. Dank seiner verteilten Architektur kann Aurora DSQL die folgenden Aktionen ausführen:

- Führen Sie DDL-Operationen als Hintergrundaufgaben aus, um Unterbrechungen zu minimieren.
- Koordinieren Sie Katalogänderungen als stark konsistente verteilte Transaktionen. Dadurch wird die atomare Transparenz über alle Knoten hinweg gewährleistet, selbst bei Ausfällen oder gleichzeitigen Vorgängen.
- Arbeiten Sie vollständig verteilt und ohne Führung in mehreren Availability Zones mit entkoppelten Rechen- und Speicherebenen.

Weitere Informationen finden Sie unter DDL und verteilte Transaktionen in Aurora DSQL.

Kompatibilität der SQL-Funktionen in Aurora DSQL

Aurora DSQL und PostgreSQL geben identische Ergebnisse für alle SQL-Abfragen zurück. Beachten Sie, dass Aurora DSQL sich von PostgreSQL ohne eine Klausel unterscheidet. `ORDER BY` In den folgenden Abschnitten erfahren Sie mehr über die Aurora-DSQL-Unterstützung für PostgreSQL-Datentypen und SQL-Befehle.

Themen

Unterstützte Datentypen in Aurora DSQL

Aurora DSQL unterstützt eine Teilmenge der gängigen PostgreSQL-Typen.

Themen

Numerische Datentypen

Aurora DSQL unterstützt die folgenden numerischen PostgreSQL-Datentypen.

Name	Aliasnamen	Reichweite und Präzision	Speichergröße	Unterstützung für Indizes
`smallint`	`int2`	-32768 bis +32767	2 Bytes	Ja
`integer`	`int`, `int4`	-2147483648 bis +2147483647	4 Bytes	Ja

Name	Aliasnamen	Reichweite und Präzision	Speichergröße	Unterstützung für Indizes
`bigint`	`int8`	-9223372036854775808 bis +9223372036854775807	8 Bytes	Ja
`real`	`float4`	Genauigkeit von 6 Dezimalziffern	4 Bytes	Ja
`double precision`	`float8`	Genauigkeit von 15 Dezimalziffern	8 Bytes	Ja

`numeric [ (`*p*`, `*s*`) ]`	`decimal [ (`*p*`, `*s*`) ]`
	
	`dec[ (`*p*`,`*s*`)]`
		Exakte Zahl mit wählbarer Genauigkeit. Die maximale Genauigkeit ist 38 und die maximale Skala ist 37. [1] Die Standardeinstellung ist`numeric (18,6)`.	8 Byte + 2 Byte pro Präzision sziffer. Die maximale Größe beträgt 27 Byte.	Nein

[1]— Wenn Sie bei der Ausführung von `CREATE TABLE` oder nicht explizit eine Größe angeben`ALTER TABLE ADD COLUMN`, erzwingt Aurora DSQL die Standardwerte. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE` OR-Anweisungen ausführen`INSERT`.

Zeichen-Datentypen

Aurora DSQL unterstützt die folgenden PostgreSQL-Zeichendatentypen.

Name	Aliasnamen	Beschreibung	Aurora DSQL-Grenze	Speichergröße	Unterstützung für Indizes
`character [ (`*n*`) ]`	`char [ (`*n*`) ]`	Zeichenfolge mit fester Länge	4096 Byte [1]	Variabel bis zu 4100 Byte	Ja

Name	Aliasname n	Beschreibung	Aurora DSQL-Grenze	Speichergröße	Unterstützung für Indizes
`character varying [ (n) ]`	`varchar [ (n) ]`	Zeichenfolge mit variabler Länge	65535 Byte 1	Variabel bis zu 65539 Byte	Ja
`bpchar [ (n) ]`		Bei fester Länge ist dies ein Alias für. `char` Bei variabler Länge ist dies ein Alias für`varchar`, wobei nachfolgende Leerzeichen semantisch unbedeutend sind.	4096 Byte (1)	Variabel bis zu 4100 Byte	Ja
`text`		Zeichenfolge mit variabler Länge	1 MiB 1	Variabel bis zu 1 MiB	Ja

1— Wenn Sie bei der Ausführung von `CREATE TABLE` oder nicht explizit eine Größe angeben`ALTER TABLE ADD COLUMN`, erzwingt Aurora DSQL die Standardwerte. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE` OR-Anweisungen ausführen`INSERT`.

Datums- und Uhrzeit-Datentypen

Aurora DSQL unterstützt die folgenden PostgreSQL-Datentypen für Datum und Uhrzeit.

Name	Aliasn n	Beschreibung	Bereich	Behebung	Speichergröße	Unterstützung für Indizes
`date`		Kalenderdatum (Jahr,	4713 V. CHR. — 5874897 N. CHR.	1 Tag	4 Bytes	Ja

Name	Aliasn	Beschreibung	Bereich	Behebung	Speichergröße	Unterstützung für Indizes
		Monat, Tag)				
`time [ (p) ] [ without time zone ]`	`times`	Tageszeit, ohne Zeitzone	0 — 1	1 Mikrosekunde	8 Bytes	Ja
`time [ (p) ] with time zone`	`timet`	Tageszeit, einschließlich Zeitzone	00:00:00 +1559 — 24:00:00 —1559	1 Mikrosekunde	12 Byte	Nein
`timestamp [ (p) ] [ without time zone ]`		Datum und Uhrzeit, ohne Zeitzone	4713 V. CHR. — 294276 N. CHR.	1 Mikrosekunde	8 Bytes	Ja
`timestamp [ (p) ] with time zone`	`timestz`	Datum und Uhrzeit, einschließlich Zeitzone	4713 V. CHR. — 294276 N. CHR.	1 Mikrosekunde	8 Bytes	Ja
`interval [ fields ] [ (p) ]`		Zeitspanne	-178000000 Jahre — 178000000 Jahre	1 Mikrosekunde	16 Byte	Nein

Verschiedene Datentypen

Aurora DSQL unterstützt die folgenden verschiedenen PostgreSQL-Datentypen.

Name	Aliasnamen	Beschreibung	Aurora DSQL-Grenze	Speichergröße	Unterstützung für Indizes
`boolean`	`bool`	Logischer/Boolescher Wert (wahr/falsch)		1 Byte	Ja
`bytea`		Binärdaten („Byte-Array“)	1 MiB [1]	Variabel bis zu einem Limit von 1 MiB	Nein
`UUID`		Universell eindeutiger Bezeichner		16 Byte	Ja

[1] — Wenn Sie bei der Ausführung von `CREATE TABLE` oder nicht explizit eine Größe angeben`ALTER TABLE ADD COLUMN`, erzwingt Aurora DSQL die Standardwerte. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE` OR-Anweisungen ausführen`INSERT`.

Abfragen von Laufzeit-Datentypen

Datentypen zur Abfragelaufzeit sind interne Datentypen, die zur Zeit der Abfrageausführung verwendet werden. Diese Typen unterscheiden sich von den PostgreSQL-kompatiblen Typen wie `varchar` und`integer`, die Sie in Ihrem Schema definieren. Stattdessen handelt es sich bei diesen Typen um Laufzeitdarstellungen, die Aurora DSQL bei der Verarbeitung einer Abfrage verwendet.

Die folgenden Datentypen werden nur während der Laufzeit der Abfrage unterstützt:

Array-Typ

Aurora DSQL unterstützt Arrays der unterstützten Datentypen. Sie können beispielsweise ein Array von ganzen Zahlen haben. Die Funktion `string_to_array` teilt eine Zeichenfolge in ein Array im PostgreSQL-Stil mit dem Kommatrennzeichen () `,` auf, wie im folgenden Beispiel gezeigt. Sie können Arrays in Ausdrücken, Funktionsausgaben oder temporären Berechnungen während der Abfrageausführung verwenden.

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

Die Funktion gibt eine Antwort zurück, die der folgenden ähnelt:

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

inet-Typ

Der Datentyp steht für IPv4 IPv6 Hostadressen und deren Subnetze. Dieser Typ ist nützlich, wenn Protokolle analysiert, nach IP-Subnetzen gefiltert oder Netzwerkberechnungen innerhalb einer Abfrage durchgeführt werden. Weitere Informationen finden Sie unter inet in der PostgreSQL-Dokumentation.

Unterstütztes SQL für Aurora DSQL

Aurora DSQL unterstützt eine Vielzahl von PostgreSQL-Kernfunktionen. In den folgenden Abschnitten erfahren Sie mehr über die allgemeine Unterstützung von PostgreSQL-Ausdrücken. Diese Liste ist nicht umfassend.

SELECT command

Aurora DSQL unterstützt die folgenden Klauseln des `SELECT` Befehls.

Primärklausel	Unterstützte Klauseln
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	

Primärklausel	Unterstützte Klauseln
USING	
WITH(allgemeine Tabellena usdrücke)	
INNER JOIN	ON
OUTER JOIN	LEFT, RIGHT, FULL, ON
CROSS JOIN	ON
UNION	ALL
INTERSECT	ALL
EXCEPT	ALL
OVER	RANK (), PARTITION BY
FOR UPDATE	

Data Definition Language (DDL)

Aurora DSQL unterstützt die folgenden PostgreSQL-DDL-Befehle.

Befehl	Primärklausel	Unterstützte Klauseln
CREATE	TABLE	Hinweise zur unterstützten Syntax des CREATE TABLE Befehls finden Sie unterCREATE TABLE.
ALTER	TABLE	Hinweise zur unterstützten Syntax des ALTER TABLE Befehls finden Sie unterALTER TABLE.
DROP	TABLE	

Befehl	Primärklausel	Unterstützte Klauseln
`CREATE`	`[UNIQUE] INDEX ASYNC`	Sie können diesen Befehl mit den folgenden Parametern verwenden:`ON`,`NULLS FIRST`,`NULLS LAST`. Hinweise zur unterstützten Syntax des `CREATE INDEX ASYNC` Befehls finden Sie unterAsynchrone Indizes in Aurora DSQL.
`DROP`	`INDEX`	
`CREATE`	`VIEW`	Weitere Hinweise zur unterstützten Syntax des `CREATE VIEW` Befehls finden Sie unter`CREATE VIEW`.
`ALTER`	`VIEW`	Hinweise zur unterstützten Syntax des `ALTER VIEW` Befehls finden Sie unterALTER VIEW.
`DROP`	`VIEW`	Hinweise zur unterstützten Syntax des `DROP VIEW` Befehls finden Sie unter`DROP VIEW`.
`CREATE`	`ROLE`, `WITH`	
`CREATE`	`FUNCTION`	`LANGUAGE SQL`
`CREATE`	`DOMAIN`	

Data Manipulation Language (DML)

Aurora DSQL unterstützt die folgenden PostgreSQL-DML-Befehle.

Befehl	Primärsatz	Unterstützte Klauseln
`INSERT`	`INTO`	`VALUES` `SELECT`

Befehl	Primärsatz	Unterstützte Klauseln
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Data Control Language (DCL)

Aurora DSQL unterstützt die folgenden PostgreSQL-DCL-Befehle.

Befehl	Unterstützte Klauseln
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Transaktionskontrollsprache (TCL)

Aurora DSQL unterstützt die folgenden PostgreSQL-TCL-Befehle.

Befehl	Unterstützte Klauseln
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

Utility-Befehle

Aurora DSQL unterstützt die folgenden PostgreSQL-Dienstprogrammbefehle:

- EXPLAIN
- ANALYZE(nur der Name der Beziehung)

Unterstützte Teilmengen von SQL-Befehlen in Aurora DSQL

Dieser PostgreSQL-Abschnitt enthält detaillierte Informationen zu unterstützten Ausdrücken, wobei der Schwerpunkt auf Befehlen mit umfangreichen Parametersätzen und Unterbefehlen liegt. CREATE TABLE in PostgreSQL bietet beispielsweise viele Klauseln und Parameter. In diesem Abschnitt werden alle PostgreSQL-Syntaxelemente beschrieben, die Aurora DSQL für diese Befehle unterstützt.

Themen

CREATE TABLE

CREATE TABLEdefiniert eine neue Tabelle.

```
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 TABLEändert die Definition einer Tabelle.

```
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 VIEWdefiniert eine neue persistente Ansicht. Aurora DSQL unterstützt keine temporären Ansichten; nur permanente Ansichten werden unterstützt.

Unterstützte Syntax

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
```

```
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Beschreibung

`CREATE VIEW`definiert eine Ansicht einer Abfrage. Die Ansicht ist nicht physisch materialisiert. Stattdessen wird die Abfrage jedes Mal ausgeführt, wenn in einer Abfrage auf die Ansicht verwiesen wird.

`CREATE or REPLACE VIEW`ist ähnlich, aber wenn bereits eine Ansicht mit demselben Namen existiert, wird sie ersetzt. Die neue Abfrage muss dieselben Spalten generieren, die von der vorhandenen Ansichtsabfrage generiert wurden (d. h. dieselben Spaltennamen in derselben Reihenfolge und mit denselben Datentypen), sie kann jedoch zusätzliche Spalten am Ende der Liste hinzufügen. Die Berechnungen, die zu den Ausgabespalten führen, können unterschiedlich sein.

Wenn ein Schemaname angegeben wird, z. B.`CREATE VIEW myschema.myview ...`), wird die Ansicht im angegebenen Schema erstellt. Andernfalls wird sie im aktuellen Schema erstellt.

Der Name der Ansicht muss sich vom Namen jeder anderen Relation (Tabelle, Index, Ansicht) im selben Schema unterscheiden.

Parameter

`CREATE VIEW`unterstützt verschiedene Parameter, um das Verhalten von automatisch aktualisierbaren Ansichten zu steuern.

`RECURSIVE`

Erzeugt eine rekursive Ansicht. Die Syntax: `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` entsprichtCREATE VIEW [schema .] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;.

Für eine rekursive Ansicht muss eine Liste mit den Namen einer Ansichtsspalte angegeben werden.

`name`

Der Name der zu erstellenden Ansicht, die optional schemaqualifiziert werden kann. Für eine rekursive Ansicht muss eine Spaltennamenliste angegeben werden.

`column_name`

Eine optionale Liste von Namen, die für Spalten der Ansicht verwendet werden sollen. Falls nicht angegeben, werden die Spaltennamen aus der Abfrage abgeleitet.

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

Diese Klausel gibt optionale Parameter für eine Ansicht an. Die folgenden Parameter werden unterstützt.

- `check_option (enum)`— Dieser Parameter kann entweder `local` oder `cascaded` sein und entspricht der Angabe`WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)`— Dieser Wert sollte verwendet werden, wenn die Ansicht Sicherheit auf Zeilenebene bieten soll. Aurora DSQL unterstützt derzeit keine Sicherheit auf Zeilenebene, aber diese Option erzwingt dennoch, dass die Bedingungen der Ansicht (und alle `WHERE` Bedingungen, die Operatoren verwenden, die als markiert sind`LEAKPROOF`) zuerst ausgewertet werden.
- `security_invoker (boolean)`— Diese Option bewirkt, dass die zugrunde liegenden Basisbeziehungen mit den Rechten des Benutzers der Ansicht und nicht des Besitzers der Ansicht verglichen werden. Vollständige Informationen finden Sie in den nachfolgenden Hinweisen.

Alle oben genannten Optionen können in vorhandenen Ansichten mit geändert werden`ALTER VIEW`.

`query`

Ein `SELECT VALUES` Or-Befehl, der die Spalten und Zeilen der Ansicht bereitstellt.

- `WITH [ CASCADED | LOCAL ] CHECK OPTION`— Diese Option steuert das Verhalten von automatisch aktualisierbaren Ansichten. Wenn diese Option angegeben ist, `INSERT` werden die `UPDATE` Befehle in der Ansicht überprüft, um sicherzustellen, dass neue Zeilen die Bedingung erfüllen, die die Ansicht definiert (das heißt, die neuen Zeilen werden überprüft, um sicherzustellen, dass sie in der Ansicht sichtbar sind). Ist dies nicht der Fall, wird die Aktualisierung abgelehnt. Wenn das nicht angegeben `CHECK OPTION` ist, `INSERT` dürfen `UPDATE` Befehle in der Ansicht Zeilen erstellen, die in der Ansicht nicht sichtbar sind. Die folgenden Prüfoptionen werden unterstützt.
- `LOCAL`— Neue Zeilen werden nur anhand der Bedingungen geprüft, die direkt in der Ansicht selbst definiert wurden. Alle Bedingungen, die in zugrunde liegenden Basisansichten definiert sind, werden nicht geprüft (es sei denn, sie spezifizieren auch die`CHECK OPTION`).

- `CASCADED`— Neue Zeilen werden anhand der Bedingungen der Ansicht und aller zugrunde liegenden Erstansichten geprüft. Wenn der angegeben `CHECK OPTION` ist und `LOCAL` weder noch angegeben `CASCADED` sind, `CASCADED` wird davon ausgegangen.

> **Note**
>
> Das `CHECK OPTION` darf nicht mit `RECURSIVE` Ansichten verwendet werden. Das `CHECK OPTION` wird nur für Ansichten unterstützt, die automatisch aktualisiert werden können.

Hinweise

Verwenden Sie die `DROP VIEW` Anweisung, um Ansichten zu löschen.

Die Namen und Datentypen der Spalten der Ansicht sollten sorgfältig geprüft werden. CREATE VIEW vista AS SELECT 'Hello World'; wird beispielsweise nicht empfohlen, da der Spaltenname standardmäßig auf eingestellt ist. `?column?;` Außerdem ist der Spaltendatentyp standardmäßig auf voreingestellt`text`, was möglicherweise nicht Ihren Wünschen entspricht.

Ein besserer Ansatz besteht darin, den Spaltennamen und den Datentyp explizit anzugeben, z. B.:`CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

Standardmäßig wird der Zugriff auf die zugrunde liegenden Basisbeziehungen, auf die in der Ansicht verwiesen wird, durch die Berechtigungen des Eigentümers der Ansicht bestimmt. In einigen Fällen kann dies verwendet werden, um einen sicheren, aber eingeschränkten Zugriff auf die zugrunde liegenden Tabellen zu ermöglichen. Allerdings sind nicht alle Ansichten manipulationssicher.

- Wenn für die Ansicht die `security_invoker` Eigenschaft auf true gesetzt ist, wird der Zugriff auf die zugrunde liegenden Basisbeziehungen durch die Berechtigungen des Benutzers bestimmt, der die Abfrage ausführt, und nicht durch den Eigentümer der Ansicht. Daher muss der Benutzer einer Sicherheitsaufruferansicht über die entsprechenden Berechtigungen für die Ansicht und die ihr zugrunde liegenden Basisbeziehungen verfügen.
- Wenn es sich bei einer der zugrunde liegenden Basisbeziehungen um eine Sicherheitsaufruferansicht handelt, wird sie so behandelt, als ob direkt von der ursprünglichen Abfrage aus auf sie zugegriffen worden wäre. Daher überprüft eine Sicherheitsaufruferansicht immer die ihr zugrunde liegenden Basisbeziehungen anhand der Berechtigungen des aktuellen Benutzers, selbst wenn auf sie von einer Ansicht ohne die `security_invoker` Eigenschaft aus zugegriffen wird.

- In der Ansicht aufgerufene Funktionen werden genauso behandelt, als ob sie direkt von der Abfrage aus aufgerufen worden wären, die die Ansicht verwendet. Daher muss der Benutzer einer Ansicht berechtigt sein, alle von der Ansicht verwendeten Funktionen aufzurufen. Funktionen in der Ansicht werden mit den Rechten des Benutzers, der die Abfrage ausführt, oder des Funktionsbesitzers ausgeführt, je nachdem, ob die Funktionen als `SECURITY INVOKER` oder definiert sind`SECURITY DEFINER`. Wenn Sie beispielsweise `CURRENT_USER` direkt in einer Ansicht aufrufen, wird immer der aufrufende Benutzer zurückgegeben, nicht der Eigentümer der Ansicht. Dies wird durch die `security_invoker` Einstellung der Ansicht nicht beeinflusst, weshalb eine Ansicht, deren Wert auf false `security_invoker` gesetzt ist, keiner `SECURITY DEFINER` Funktion entspricht.
- Der Benutzer, der eine Ansicht erstellt oder ersetzt, muss über `USAGE` Berechtigungen für alle Schemas verfügen, auf die in der Ansichtsabfrage verwiesen wird, um die referenzierten Objekte in diesen Schemas nachschlagen zu können. Beachten Sie jedoch, dass diese Suche nur stattfindet, wenn die Ansicht erstellt oder ersetzt wird. Daher benötigt der Benutzer der Ansicht nur die `USAGE` Berechtigung für das Schema, das die Ansicht enthält, nicht für die Schemas, auf die in der View-Abfrage verwiesen wird, auch nicht für eine Sicherheitsaufruf-Ansicht.
- Wenn in einer vorhandenen Ansicht verwendet `CREATE OR REPLACE VIEW` wird, werden nur die definierende `SELECT` Regel der Ansicht sowie alle `WITH ( ... )` Parameter und deren `CHECK OPTION` Werte geändert. Andere Eigenschaften der Ansicht, einschließlich Besitzrechte, Berechtigungen und Regeln, bei denen es sich nicht um Select-Regeln handelt, bleiben unverändert. Sie müssen Eigentümer der Ansicht sein, um sie ersetzen zu können (dazu gehört auch, dass Sie Mitglied der Eigentümerrolle sind).

Aktualisierbare Ansichten

Einfache Ansichten sind automatisch aktualisierbar: Das System ermöglicht die Verwendung von `INSERT` `UPDATE` ,- und `DELETE` Anweisungen in der Ansicht auf die gleiche Weise wie in einer normalen Tabelle. Eine Ansicht ist automatisch aktualisierbar, wenn sie alle der folgenden Bedingungen erfüllt:

- Die Ansicht muss genau einen Eintrag in ihrer `FROM` Liste enthalten, bei dem es sich um eine Tabelle oder eine andere aktualisierbare Ansicht handeln muss.
- Die Sichtdefinition darf keine`WITH`,,`DISTINCT`, `GROUP BY` `HAVING``LIMIT`, oder `OFFSET` Klauseln auf der obersten Ebene enthalten.
- Die Ansichtsdefinition darf keine Mengenoperationen (`UNION``INTERSECT`, oder`EXCEPT`) auf der obersten Ebene enthalten.

- Die Auswahlliste der Ansicht darf keine Aggregate, Fensterfunktionen oder Funktionen zur Rückgabe von Mengen enthalten.

Eine automatisch aktualisierbare Ansicht kann eine Mischung aus aktualisierbaren und nicht aktualisierbaren Spalten enthalten. Eine Spalte ist aktualisierbar, wenn es sich um einen einfachen Verweis auf eine aktualisierbare Spalte der zugrunde liegenden Basisrelation handelt. Andernfalls ist die Spalte schreibgeschützt, und es tritt ein Fehler auf, wenn eine `INSERT` `UPDATE` Oder-Anweisung versucht, ihr einen Wert zuzuweisen.

Bei automatisch aktualisierbaren Ansichten konvertiert das System jede`INSERT`,`UPDATE`, `DELETE` - Anweisung in der Ansicht in die entsprechende Anweisung in der zugrunde liegenden Basisrelation. `INSERT`Anweisungen mit einer `ON CONFLICT UPDATE` Klausel werden vollständig unterstützt.

Wenn eine automatisch aktualisierbare Ansicht eine `WHERE` Bedingung enthält, schränkt die Bedingung ein, welche Zeilen der Basisrelation in der Ansicht geändert werden können `UPDATE` und welche `DELETE` Anweisungen in der Ansicht enthalten sind. Eine Zeile `UPDATE` kann jedoch so geändert werden, dass sie die `WHERE` Bedingung nicht mehr erfüllt, sodass sie in der Ansicht unsichtbar wird. In ähnlicher Weise kann ein `INSERT` Befehl möglicherweise Zeilen mit Basisbeziehungen einfügen, die die `WHERE` Bedingung nicht erfüllen, sodass sie in der Ansicht unsichtbar sind. `ON CONFLICT UPDATE`kann sich in ähnlicher Weise auf eine vorhandene Zeile auswirken, die in der Ansicht nicht sichtbar ist.

Sie können die `UPDATE` Befehle verwenden`CHECK OPTION`, um `INSERT` zu verhindern, dass Zeilen erstellt werden, die in der Ansicht nicht sichtbar sind.

Wenn eine automatisch aktualisierbare Ansicht mit der Eigenschaft security_barrier gekennzeichnet ist, werden alle Bedingungen der Ansicht (und alle `WHERE` Bedingungen, die Operatoren verwenden, die als markiert sind`LEAKPROOF`) immer vor allen Bedingungen ausgewertet, die ein Benutzer der Ansicht hinzugefügt hat. Beachten Sie, dass aus diesem Grund Zeilen, die letztendlich nicht zurückgegeben werden (weil sie die `WHERE` Bedingungen des Benutzers nicht erfüllen), trotzdem gesperrt werden können. Sie können `EXPLAIN` damit sehen, welche Bedingungen auf der Beziehungsebene angewendet werden (und daher keine Zeilen sperren) und welche nicht.

Eine komplexere Ansicht, die nicht all diese Bedingungen erfüllt, ist standardmäßig schreibgeschützt: Das System erlaubt kein Einfügen, Aktualisieren oder Löschen in der Ansicht.

> **Note**
>
> Der Benutzer, der das Einfügen, Aktualisieren oder Löschen in der Ansicht ausführt, muss über die entsprechende Berechtigung zum Einfügen, Aktualisieren oder Löschen für die Ansicht verfügen. Standardmäßig muss der Besitzer der Ansicht über die entsprechenden Rechte für die zugrunde liegenden Basisbeziehungen verfügen, während der Benutzer, der die Aktualisierung durchführt, keine Berechtigungen für die zugrunde liegenden Basisbeziehungen benötigt. Wenn security_invoker für die Ansicht jedoch auf true gesetzt ist, muss der Benutzer, der das Update durchführt, und nicht der Eigentümer der Ansicht, über die entsprechenden Rechte für die zugrunde liegenden Basisbeziehungen verfügen.

Beispiele

Um eine Ansicht zu erstellen, die aus allen Comedy-Filmen besteht.

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

Dadurch wird eine Ansicht erstellt, die die Spalten enthält, die sich zum Zeitpunkt der Erstellung der Ansicht in der `film` Tabelle befanden. *Es wurde zwar verwendet, um die Ansicht zu erstellen, Spalten, die später zur Tabelle hinzugefügt wurden, sind jedoch nicht Teil der Ansicht.

Erstellen Sie eine Ansicht mit`LOCAL CHECK OPTION`.

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

Dadurch wird eine Ansicht erstellt, die `kind` sowohl die als auch `classification` die neuen Zeilen überprüft.

Erstellen Sie eine Ansicht mit einer Mischung aus aktualisierbaren und nicht aktualisierbaren Spalten.

```
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';
```

Diese Ansicht unterstützt,`INSERT`, und`UPDATE`. `DELETE` Alle Spalten aus der Filmtabelle können aktualisiert werden, wohingegen die berechneten Spalten `country` und E schreibgeschützt `avg_rating` sind.

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

> **Note**
>
> Obwohl der Name der rekursiven Ansicht in diesem `CREATE` Fall schemaqualifiziert ist, ist ihr interner Selbstreferenz nicht schemaqualifiziert. Das liegt daran, dass der Name des implizit erstellten Common Table Expression (CTE) nicht schemaqualifiziert werden kann.

Kompatibilität

`CREATE OR REPLACE VIEW`ist eine PostgreSQL-Spracherweiterung. Die `WITH ( ... )` Klausel ist ebenfalls eine Erweiterung, ebenso wie Sicherheitsbarrierenansichten und Sicherheitsaufruferansichten. Aurora DSQL unterstützt diese Spracherweiterungen.

ALTER VIEW

Die `ALTER VIEW` Anweisung ermöglicht das Ändern verschiedener Eigenschaften einer vorhandenen Ansicht, und Aurora DSQL unterstützt die gesamte PostgreSQL-Syntax für diesen Befehl.

Unterstützte Syntax

```
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 [, ... ] )
```

Beschreibung

`ALTER VIEW`ändert verschiedene Hilfseigenschaften einer Ansicht. (Wenn Sie die definierende Abfrage der Ansicht ändern möchten, verwenden Sie`CREATE OR REPLACE VIEW`.) Sie müssen Eigentümer der Ansicht sein, um sie verwenden zu können`ALTER VIEW`. Um das Schema einer Ansicht zu ändern, müssen Sie auch über `CREATE` Berechtigungen für das neue Schema verfügen. Um den Besitzer zu ändern, müssen Sie in der Lage sein, `SET ROLE` auf die neue Eigentümerrolle zuzugreifen, und diese Rolle muss über `CREATE` Berechtigungen für das Schema der Ansicht verfügen. Diese Einschränkungen legen fest, dass das Ändern des Besitzers nichts bewirkt, was Sie nicht tun könnten, indem Sie die Ansicht löschen und neu erstellen.)

Parameter

`ALTER VIEW`-Parameter

`name`

Der Name (optional schemaqualifiziert) einer vorhandenen Ansicht.

`column_name`

Neuer Name für eine bestehende Spalte.

`IF EXISTS`

Geben Sie keinen Fehler aus, wenn die Ansicht nicht existiert. In diesem Fall wird eine Mitteilung herausgegeben.

`SET/DROP DEFAULT`

Mit diesen Formularen wird der Standardwert für eine Spalte festgelegt oder entfernt. Der Standardwert für eine Ansichtsspalte wird durch einen beliebigen `INSERT` `UPDATE` OR-Befehl ersetzt, bei dem das Ziel die Ansicht ist. Der Standardwert für die Ansicht hat Vorrang vor allen Standardwerten aus zugrunde liegenden Beziehungen.

new_owner

Der Benutzername des neuen Besitzers der Ansicht.

new_name

Der neue Name für die Ansicht.

neues_Schema

Das neue Schema für die Ansicht.

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

Legt eine Ansichtsoption fest oder setzt sie zurück. Die folgenden Optionen werden unterstützt.

- `check_option (enum)`- Ändert die Checkoption der Ansicht. Der Wert muss `local` oder `cascaded` sein.
- `security_barrier (boolean)`- Ändert die Sicherheitsbarriere-Eigenschaft der Ansicht. Der Wert muss ein boolescher Wert sein, z. B. oder. `true` `false`
- `security_invoker (boolean)`- Ändert die Sicherheitsbarriere-Eigenschaft der Ansicht. Der Wert muss ein boolescher Wert sein, z. B. oder. `true` `false`

Hinweise

`ALTER TABLE`Kann aus historischen PostgreSQL-Gründen auch mit Ansichten verwendet werden; aber die einzigen Varianten`ALTER TABLE`, die mit Ansichten erlaubt sind, entsprechen den zuvor gezeigten.

Beispiele

Umbenennen der Ansicht in. `foo` `bar`

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

Einer aktualisierbaren Ansicht einen Standardspaltenwert zuordnen.

```
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
```

Kompatibilität

`ALTER VIEW`ist eine PostgreSQL-Erweiterung des SQL-Standards, den Aurora DSQL unterstützt.

DROP VIEW

Die `DROP VIEW` Anweisung entfernt eine vorhandene Ansicht. Aurora DSQL unterstützt die vollständige PostgreSQL-Syntax für diesen Befehl.

Unterstützte Syntax

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

Beschreibung

`DROP VIEW`löscht eine bestehende Ansicht. Um diesen Befehl ausführen zu können, müssen Sie der Besitzer der Ansicht sein.

Parameter

`IF EXISTS`

Geben Sie keinen Fehler aus, wenn die Ansicht nicht existiert. In diesem Fall wird eine Mitteilung herausgegeben.

`name`

Der Name (optional schemaqualifiziert) der Ansicht, die entfernt werden soll.

`CASCADE`

Löscht automatisch Objekte, die von der Ansicht abhängen (z. B. andere Ansichten), und wiederum alle Objekte, die von diesen Objekten abhängen.

`RESTRICT`

Lehnen Sie es ab, die Ansicht zu löschen, wenn Objekte davon abhängen. Dies ist die Standardeinstellung.

Beispiele

```
DROP VIEW kinds;
```

Kompatibilität

Dieser Befehl entspricht dem SQL-Standard, mit der Ausnahme, dass der Standard nur das Löschen einer Ansicht pro Befehl zulässt, abgesehen von der `IF EXISTS` Option, einer PostgreSQL-Erweiterung, die Aurora DSQL unterstützt.

Nicht unterstützte PostgreSQL-Funktionen in Aurora DSQL

Aurora DSQL ist PostgreSQL-kompatibel. Das bedeutet, dass Aurora DSQL relationale Kernfunktionen wie ACID-Transaktionen, Sekundärindizes, Joins, Insert und Updates unterstützt. Einen Überblick über die unterstützten SQL-Funktionen finden Sie unter Unterstützte SQL-Ausdrücke.

In den folgenden Abschnitten wird hervorgehoben, welche PostgreSQL-Funktionen derzeit in Aurora DSQL nicht unterstützt werden.

Nicht unterstützte Objekte

Zu den Objekten, die von Aurora DSQL nicht unterstützt werden, gehören:

- Mehrere Datenbanken auf einem einzigen Aurora DSQL-Cluster
- Temporäre Tabellen
- Auslöser
- Typen (teilweise Unterstützung)
- Tablespaces
- Funktionen, die in anderen Sprachen als SQL geschrieben wurden
- Sequenzen
- Partitionen

Einschränkungen werden nicht unterstützt

- Fremdschlüssel
- Ausschluss-Einschränkungen

Nicht unterstützte Befehle

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

> **Note**
>
> Aurora SQL erfordert kein Staubsaugen. Das System verwaltet Statistiken und verwaltet die Speicheroptimierung automatisch ohne manuelle Vakuumbefehle.

Erweiterungen werden nicht unterstützt

Aurora DSQL unterstützt keine PostgreSQL-Erweiterungen. Die folgende Tabelle zeigt Erweiterungen, die nicht unterstützt werden:

- `PL/pgSQL`
- `PostGIS`
- `PGVector`
- `PGAudit`
- `Postgres_FDW`
- `PGCron`
- `pg_stat_statements`

Nicht unterstützte SQL-Ausdrücke

In der folgenden Tabelle werden Klauseln beschrieben, die in Aurora DSQL nicht unterstützt werden.

Kategorie	Primäre Klausel	Klausel wird nicht unterstützt
`CREATE`	`INDEX ASYNC`	`ASC | DESC`
`CREATE`	`INDEX`[1]	

Kategorie	Primäre Klausel	Klausel wird nicht unterstützt
TRUNCATE		
ALTER	SYSTEM	Alle ALTER SYSTEM Befehle sind blockiert.
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION
CREATE	FUNCTION	LANGUAGE *non-sql-lang* , wo *non-sql-lang* ist eine andere Sprache als SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	Sie können keine zusätzlichen Datenbanken erstellen.

[1] Informationen Asynchrone Indizes in Aurora DSQL zum Erstellen eines Indexes für eine Spalte einer angegebenen Tabelle finden Sie unter.

Überlegungen zu Aurora DSQL zur PostgreSQL-Kompatibilität

Beachten Sie bei der Verwendung von Aurora DSQL die folgenden Kompatibilitätseinschränkungen. Allgemeine Überlegungen finden Sie unterÜberlegungen zur Arbeit mit Amazon Aurora DSQL. Informationen zu Kontingenten und Beschränkungen finden Sie unterCluster-Kontingente und Datenbank-Limits in Amazon Aurora DSQL.

- Aurora DSQL verwendet eine einzige integrierte Datenbank mit dem Namen`postgres`. Sie können keine zusätzlichen Datenbanken erstellen oder die `postgres` Datenbank umbenennen oder löschen.
- Die `postgres` Datenbank verwendet die UTF-8-Zeichenkodierung. Sie können die Kodierung nicht ändern.
- Die Datenbank verwendet nur die `C` Sortierung.
- Aurora DSQL verwendet `UTC` als Systemzeitzone. Sie können die Zeitzone nicht mit Parametern oder SQL-Anweisungen wie ändern. `SET TIMEZONE`
- Die Transaktionsisolationsstufe ist bei PostgreSQL `Repeatable Read` festgelegt.
- Für Transaktionen gelten die folgenden Einschränkungen:
  - Eine Transaktion kann DDL- und DML-Operationen nicht kombinieren
  - Eine Transaktion kann nur eine DDL-Anweisung enthalten
  - Eine Transaktion kann unabhängig von der Anzahl der Sekundärindizes bis zu 3.000 Zeilen ändern
  - Die Obergrenze von 3.000 Zeilen gilt für alle DML-Anweisungen (,,) `INSERT` `UPDATE` `DELETE`
- Bei Datenbankverbindungen wird das Timeout nach 1 Stunde überschritten.
- Aurora DSQL lässt Sie derzeit nicht ausführen`GRANT [permission] ON DATABASE`. Wenn Sie versuchen, diese Anweisung auszuführen, gibt Aurora DSQL die Fehlermeldung `ERROR: unsupported object type in GRANT` zurück.
- Aurora DSQL lässt nicht zu, dass Benutzerrollen ohne Administratorrechte den `CREATE SCHEMA` Befehl ausführen. Sie können den `GRANT [permission] on DATABASE` Befehl nicht ausführen und `CREATE` Berechtigungen für die Datenbank gewähren. Wenn eine Benutzerrolle ohne Administratorrechte versucht, ein Schema zu erstellen, kehrt Aurora DSQL mit der Fehlermeldung zurück. `ERROR: permission denied for database postgres`
- Benutzer ohne Administratorrechte können keine Objekte im öffentlichen Schema erstellen. Nur Admin-Benutzer können Objekte im öffentlichen Schema erstellen. Die Admin-Benutzerrolle ist berechtigt, Benutzern ohne Administratorrechte Lese-, Schreib- und Änderungszugriff auf diese Objekte zu gewähren, sie kann jedoch keine `CREATE` Berechtigungen für das öffentliche Schema selbst gewähren. Benutzer ohne Administratorrechte müssen unterschiedliche, vom Benutzer erstellte Schemas für die Objekterstellung verwenden.
- Aurora DSQL unterstützt den Befehl `ALTER ROLE [] CONNECTION LIMIT` nicht. Wenden Sie sich an den AWS Support, wenn Sie eine Erhöhung des Verbindungslimits benötigen.

- Aurora DSQL unterstützt asyncpg, den asynchronen PostgreSQL-Datenbanktreiber für Python, nicht.

Parallelitätssteuerung in Aurora DSQL

Durch Parallelität können mehrere Sitzungen gleichzeitig auf Daten zugreifen und diese ändern, ohne die Datenintegrität und -konsistenz zu beeinträchtigen. Aurora DSQL bietet PostgreSQL-Kompatibilität und implementiert gleichzeitig einen modernen, sperrfreien Parallelitätskontrollmechanismus. Es gewährleistet die vollständige ACID-Konformität durch Snapshot-Isolierung und gewährleistet so Datenkonsistenz und Zuverlässigkeit.

Ein entscheidender Vorteil von Aurora DSQL ist die lockfreie Architektur, die häufig auftretende Engpässe bei der Datenbankleistung beseitigt. Aurora DSQL verhindert, dass langsame Transaktionen andere Operationen blockieren, und beseitigt das Risiko von Deadlocks. Dieser Ansatz macht Aurora DSQL besonders wertvoll für Anwendungen mit hohem Durchsatz, bei denen Leistung und Skalierbarkeit entscheidend sind.

Transaktionskonflikte

Aurora DSQL verwendet optimistische Parallelitätssteuerung (OCC), die anders funktioniert als herkömmliche sperrenbasierte Systeme. Anstatt Sperren zu verwenden, bewertet OCC Konflikte beim Festschreiben. Wenn mehrere Transaktionen beim Aktualisieren derselben Zeile in Konflikt geraten, verwaltet Aurora DSQL Transaktionen wie folgt:

- Die Transaktion mit der frühesten Commit-Zeit wird von Aurora DSQL verarbeitet.
- Bei widersprüchlichen Transaktionen wird ein PostgreSQL-Serialisierungsfehler angezeigt, der darauf hinweist, dass ein erneuter Versuch erforderlich ist.

Entwerfen Sie Ihre Anwendungen so, dass sie Wiederholungslogik implementieren, um Konflikte zu behandeln. Das ideale Entwurfsmuster ist idempotent und ermöglicht, wann immer möglich, die Wiederholung von Transaktionen als erste Möglichkeit. Die empfohlene Logik ähnelt der Abbruch- und Wiederholungslogik in einer standardmäßigen PostgreSQL-Sperrtimeout- oder Deadlock-Situation. OCC erfordert jedoch, dass Ihre Anwendungen diese Logik häufiger anwenden.

Richtlinien für die Optimierung der Transaktionsleistung

Um die Leistung zu optimieren, sollten Sie starke Konflikte bei einzelnen Schlüsseln oder kleinen Schlüsselbereichen minimieren. Um dieses Ziel zu erreichen, sollten Sie Ihr Schema so entwerfen,

dass Updates über den gesamten Cluster-Schlüsselbereich verteilt werden. Beachten Sie dabei die folgenden Richtlinien:

- Wählen Sie einen zufälligen Primärschlüssel für Ihre Tabellen.
- Vermeiden Sie Muster, die zu Konflikten bei einzelnen Schlüsseln führen. Dieser Ansatz gewährleistet eine optimale Leistung auch bei steigendem Transaktionsvolumen.

DDL und verteilte Transaktionen in Aurora DSQL

Die Datendefinitionssprache (DDL) verhält sich in Aurora DSQL anders als in PostgreSQL. Aurora DSQL verfügt über eine verteilte Multi-AZ-Datenbankschicht und Shared-Nothing-Datenbankschicht, die auf Mehrmandanten-Rechen- und Speicherflotten aufbaut. Da kein einziger primärer Datenbankknoten oder Leader existiert, ist der Datenbankkatalog verteilt. Somit verwaltet Aurora DSQL DDL-Schemaänderungen als verteilte Transaktionen.

Insbesondere verhält sich DDL in Aurora DSQL wie folgt anders:

Fehler bei der Parallelitätssteuerung

Aurora DSQL gibt einen Fehler wegen Verletzung der Parallelitätskontrolle zurück, wenn Sie eine Transaktion ausführen, während eine andere Transaktion eine Ressource aktualisiert. Stellen Sie sich zum Beispiel die folgende Abfolge von Aktionen vor:

1. In Sitzung 1 fügt ein Benutzer der Tabelle eine Spalte hinzu`mytable`.
2. In Sitzung 2 versucht ein Benutzer, eine Zeile in einzufügen`mytable`.

   Aurora DSQL gibt den Fehler zurück `SQL Error [40001]: ERROR: schema has been updated by another transaction, please retry: (OC001).`

DDL und DML in derselben Transaktion

Transaktionen in Aurora DSQL können nur eine DDL-Anweisung enthalten und nicht gleichzeitig DDL- und DML-Anweisungen enthalten. Diese Einschränkung bedeutet, dass Sie innerhalb derselben Transaktion keine Tabelle erstellen und Daten in dieselbe Tabelle einfügen können. Aurora DSQL unterstützt beispielsweise die folgenden sequentiellen Transaktionen.

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

```
COMMIT;

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

Aurora DSQL unterstützt die folgende Transaktion nicht, die `CREATE` sowohl als auch `INSERT` Anweisungen enthält.

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

Asynchrone DDL

In Standard-PostgreSQL sperren DDL-Operationen wie das `CREATE INDEX` Sperren der betroffenen Tabelle, sodass sie für Lese- und Schreibvorgänge aus anderen Sitzungen nicht verfügbar ist. In Aurora DSQL werden diese DDL-Anweisungen asynchron mithilfe eines Hintergrundmanagers ausgeführt. Der Zugriff auf die betroffene Tabelle ist nicht blockiert. Somit kann DDL auf großen Tabellen ohne Ausfallzeiten oder Leistungseinbußen ausgeführt werden. Weitere Informationen zum asynchronen Job-Manager in Aurora DSQL finden Sie unter. Asynchrone Indizes in Aurora DSQL

Primärschlüssel in Aurora DSQL

In Aurora DSQL ist ein Primärschlüssel eine Funktion, die Tabellendaten physisch organisiert. Es ähnelt der `CLUSTER` Operation in PostgreSQL oder einem geclusterten Index in anderen Datenbanken. Wenn Sie einen Primärschlüssel definieren, erstellt Aurora DSQL einen Index, der alle Spalten in der Tabelle enthält. Die Primärschlüsselstruktur in Aurora DSQL gewährleistet einen effizienten Datenzugriff und eine effiziente Verwaltung.

Datenstruktur und Speicherung

Wenn Sie einen Primärschlüssel definieren, speichert Aurora DSQL Tabellendaten in der Reihenfolge der Primärschlüssel. Diese indexorganisierte Struktur ermöglicht eine Primärschlüsselsuche, bei der alle Spaltenwerte direkt abgerufen werden können, anstatt wie bei einem herkömmlichen B-Tree-Index einem Zeiger auf die Daten zu folgen. Im Gegensatz zur `CLUSTER` Operation in

PostgreSQL, bei der Daten nur einmal reorganisiert werden, behält Aurora DSQL diese Reihenfolge automatisch und kontinuierlich bei. Dieser Ansatz verbessert die Leistung von Abfragen, die auf Primärschlüsselzugriff angewiesen sind.

Aurora DSQL verwendet den Primärschlüssel auch, um einen clusterweiten eindeutigen Schlüssel für jede Zeile in Tabellen und Indizes zu generieren. Dieser eindeutige Schlüssel unterstützt auch die verteilte Datenverwaltung. Er ermöglicht die automatische Partitionierung von Daten über mehrere Knoten hinweg und unterstützt skalierbaren Speicher und hohe Parallelität. Infolgedessen hilft die Primärschlüsselstruktur Aurora DSQL dabei, automatisch zu skalieren und gleichzeitige Workloads effizient zu verwalten.

Richtlinien für die Auswahl eines Primärschlüssels

Beachten Sie bei der Auswahl und Verwendung eines Primärschlüssels in Aurora DSQL die folgenden Richtlinien:

- Definieren Sie einen Primärschlüssel, wenn Sie eine Tabelle erstellen. Sie können diesen Schlüssel später nicht ändern oder einen neuen Primärschlüssel hinzufügen. Der Primärschlüssel wird Teil des clusterweiten Schlüssels, der für die Datenpartitionierung und die automatische Skalierung des Schreibdurchsatzes verwendet wird. Wenn Sie keinen Primärschlüssel angeben, weist Aurora DSQL eine synthetische versteckte ID zu.
- Vermeiden Sie bei Tabellen mit hohem Schreibvolumen die Verwendung monoton steigender Ganzzahlen als Primärschlüssel. Dies kann zu Leistungseinbußen führen, wenn alle neuen Einfügungen auf eine einzige Partition geleitet werden. Verwenden Sie stattdessen Primärschlüssel mit zufälliger Verteilung, um eine gleichmäßige Verteilung der Schreibvorgänge auf die Speicherpartitionen sicherzustellen.
- Für Tabellen, die sich selten ändern oder schreibgeschützt sind, können Sie einen aufsteigenden Schlüssel verwenden. Beispiele für aufsteigende Schlüssel sind Zeitstempel oder Sequenznummern. Ein dichter Schlüssel hat viele eng beieinander liegende oder doppelte Werte. Sie können einen aufsteigenden Schlüssel verwenden, auch wenn er dicht ist, da die Schreibleistung weniger wichtig ist.
- Wenn ein vollständiger Tabellenscan Ihre Leistungsanforderungen nicht erfüllt, wählen Sie eine effizientere Zugriffsmethode. In den meisten Fällen bedeutet dies, dass Sie einen Primärschlüssel verwenden, der Ihrem häufigsten Join- und Lookup-Schlüssel in Abfragen entspricht.
- Die maximale kombinierte Größe von Spalten in einem Primärschlüssel beträgt 1 Kibibyte. Weitere Informationen finden Sie unter Datenbanklimits in Aurora DSQL und Unterstützte Datentypen in Aurora DSQL.

- Sie können bis zu 8 Spalten in einen Primärschlüssel oder einen Sekundärindex aufnehmen. Weitere Informationen finden Sie unter Datenbanklimits in Aurora DSQL und Unterstützte Datentypen in Aurora DSQL.

Asynchrone Indizes in Aurora DSQL

Der `CREATE INDEX ASYNC` Befehl erstellt einen Index für eine oder mehrere Spalten einer angegebenen Tabelle. Bei diesem Befehl handelt es sich um eine asynchrone DDL-Operation, die andere Transaktionen nicht blockiert. Wenn Sie es ausführen`CREATE INDEX ASYNC`, gibt Aurora DSQL sofort a `job_id` zurück.

Sie können den Status dieses asynchronen Jobs mithilfe der `sys.jobs` Systemansicht überwachen. Während der Indexerstellung können Sie die folgenden Verfahren und Befehle verwenden:

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

Blockiert die aktuelle Sitzung, bis der angegebene Job abgeschlossen ist oder fehlschlägt. Gibt einen booleschen Wert zurück, der auf Erfolg oder Misserfolg hinweist.

`DROP INDEX`

Bricht einen laufenden Indexerstellungsauftrag ab.

Wenn die asynchrone Indexerstellung abgeschlossen ist, aktualisiert Aurora DSQL den Systemkatalog, um den Index als aktiv zu kennzeichnen.

> **Note**
>
> Beachten Sie, dass bei gleichzeitigen Transaktionen, die während dieser Aktualisierung auf Objekte im selben Namespace zugreifen, Parallelitätsfehler auftreten können.

Wenn Aurora DSQL eine asynchrone Indexaufgabe beendet, aktualisiert es den Systemkatalog, um anzuzeigen, dass der Index aktiv ist. Wenn andere Transaktionen zu diesem Zeitpunkt auf die Objekte im selben Namespace verweisen, wird möglicherweise ein Parallelitätsfehler angezeigt.

Syntax

`CREATE INDEX ASYNC`verwendet die folgende Syntax.

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

Parameter

`UNIQUE`

Weist Aurora DSQL an, bei der Indexerstellung und bei jedem Hinzufügen von Daten nach doppelten Werten in der Tabelle zu suchen. Wenn Sie diesen Parameter angeben, wird bei Einfüge- und Aktualisierungsvorgängen, die zu doppelten Einträgen führen würden, ein Fehler generiert.

`IF NOT EXISTS`

Zeigt an, dass Aurora DSQL keine Ausnahme auslösen sollte, wenn bereits ein Index mit demselben Namen existiert. In dieser Situation erstellt Aurora DSQL den neuen Index nicht. Beachten Sie, dass der Index, den Sie zu erstellen versuchen, eine ganz andere Struktur als der bestehende Index haben kann. Wenn Sie diesen Parameter angeben, ist der Indexname erforderlich.

`name`

Der Indexname. Sie können den Namen Ihres Schemas nicht in diesen Parameter aufnehmen.

Aurora DSQL erstellt den Index im gleichen Schema wie die übergeordnete Tabelle. Der Name des Indexes muss sich vom Namen jedes anderen Objekts, z. B. einer Tabelle oder eines Indexes, im Schema unterscheiden.

Wenn Sie keinen Namen angeben, generiert Aurora DSQL automatisch einen Namen, der auf dem Namen der übergeordneten Tabelle und der indizierten Spalte basiert. Wenn Sie beispielsweise ausführen`CREATE INDEX ASYNC on table1 (col1, col2)`, benennt Aurora DSQL den Index `table1_col1_col2_idx` automatisch.

`NULLS FIRST | LAST`

Die Sortierreihenfolge von Nullspalten und Spalten, die nicht Null sind. `FIRST`gibt an, dass Aurora DSQL Nullspalten vor Nicht-Null-Spalten sortieren sollte. `LAST`gibt an, dass Aurora DSQL Nullspalten nach Nicht-Null-Spalten sortieren soll.

`INCLUDE`

Eine Liste von Spalten, die als Nicht-Schlüsselspalten in den Index aufgenommen werden sollen. Sie können keine Nichtschlüsselspalte in einer Indexscan-Suchqualifikation verwenden. Aurora DSQL ignoriert die Spalte im Hinblick auf die Eindeutigkeit eines Indexes.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Gibt an, ob Aurora DSQL Nullwerte in einem eindeutigen Index als unterschiedlich betrachten soll. Die Standardeinstellung ist`DISTINCT`, was bedeutet, dass ein eindeutiger Index mehrere Nullwerte in einer Spalte enthalten kann. `NOT DISTINCT`gibt an, dass ein Index nicht mehrere Nullwerte in einer Spalte enthalten kann.

Nutzungshinweise

Berücksichtigen Sie die folgenden Hinweise:

- Der `CREATE INDEX ASYNC` Befehl führt keine Sperren ein. Es hat auch keinen Einfluss auf die Basistabelle, die Aurora DSQL verwendet, um den Index zu erstellen.
- Bei Schemamigrationsvorgängen ist das `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* Verfahren nützlich. Es stellt sicher, dass nachfolgende DDL- und DML-Operationen auf den neu erstellten Index abzielen.
- Jedes Mal, wenn Aurora DSQL eine neue asynchrone Aufgabe ausführt, überprüft es die `sys.jobs` Ansicht und löscht Aufgaben, die einen Status von `completed` oder länger als `failed` 30 Minuten haben. Zeigt also `sys.jobs` hauptsächlich Aufgaben an, die gerade bearbeitet werden, und enthält keine Informationen über alte Aufgaben.
- Wenn Aurora DSQL keinen asynchronen Index erstellen kann, bleibt der Index erhalten. `INVALID` Bei eindeutigen Indizes unterliegen DML-Operationen Eindeutigkeitsbeschränkungen, bis Sie den Index löschen. Es wird empfohlen, ungültige Indizes zu löschen und sie neu zu erstellen.

Einen Index erstellen: Beispiel

Das folgende Beispiel zeigt, wie Sie ein Schema, eine Tabelle und dann einen Index erstellen.

1. Erstellen Sie eine Tabelle mit dem Namen „`test.departments`“.

```
CREATE SCHEMA test;

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

2. Fügt eine Zeile in die Tabelle ein.

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

3. Erstellen Sie einen asynchronen Index.

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

   Der `CREATE INDEX` Befehl gibt eine Job-ID zurück, wie unten gezeigt.

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

   Das `job_id` zeigt an, dass Aurora DSQL einen neuen Job zur Indexerstellung eingereicht hat. Sie können das Verfahren verwendensys.wait_for_job(job_id)*'your_index_creation_job_id'*, um andere Arbeiten an der Sitzung zu blockieren, bis der Job abgeschlossen ist oder das Timeout überschritten wird.

Den Status der Indexerstellung abfragen: Beispiel

Fragen Sie die `sys.jobs` Systemansicht ab, um den Erstellungsstatus Ihres Indexes zu überprüfen, wie im folgenden Beispiel gezeigt.

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

Aurora DSQL gibt eine Antwort ähnlich der folgenden zurück.

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

Die Statusspalte kann einen der folgenden Werte haben.

submitted	**processing**	**failed**	**completed**
Die Aufgabe wurde eingereicht, aber Aurora DSQL hat noch nicht begonnen, sie zu verarbeiten.	Aurora DSQL verarbeitet die Aufgabe.	Die Aufgabe ist fehlgeschlagen. Weitere Informati onen finden Sie in der Detailspalte. Wenn Aurora DSQL den Index nicht erstellen konnte, entfernt Aurora DSQL die Indexdefinition nicht automatisch. Sie müssen den Index manuell mit dem `DROP INDEX` Befehl entfernen.	Aurora SQL

Sie können den Status des Indexes auch über die Katalogtabellen `pg_index` und `pg_class` abfragen. Insbesondere die Attribute `indisvalid` und `indisimmediate` können Ihnen sagen, in welchem Zustand sich Ihr Index befindet. Aurora DSQL erstellt zwar Ihren Index, hat aber den Anfangsstatus. `INVALID` Das `indisvalid` Kennzeichen für den Index gibt `FALSE` oder zurück`f`, was darauf hinweist, dass der Index nicht gültig ist. Wenn das Flag `TRUE` oder zurückgibt`t`, ist der Index bereit.

```
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)
```

Fehler bei der Erstellung eines eindeutigen Indexes

Wenn Ihr asynchroner Auftrag zur Erstellung eines eindeutigen Indexes den Status „Fehlgeschlagen“ mit den entsprechenden Details anzeigt`Found duplicate key while validating index for UCVs`, deutet dies darauf hin, dass ein eindeutiger Index aufgrund von Verstößen gegen Eindeutigkeitsbeschränkungen nicht erstellt werden konnte.

Um Fehler bei der Erstellung eines eindeutigen Indexes zu beheben

1. Entfernen Sie alle Zeilen in Ihrer Primärtabelle, die doppelte Einträge für die in Ihrem eindeutigen sekundären Index angegebenen Schlüssel enthalten.
2. Löscht den fehlgeschlagenen Index.
3. Geben Sie einen neuen Befehl zum Erstellen eines Index ein.

Erkennung von Eindeutigkeitsverletzungen in Primärtabellen

Die folgende SQL-Abfrage hilft Ihnen dabei, doppelte Werte in einer bestimmten Spalte Ihrer Tabelle zu identifizieren. Dies ist besonders nützlich, wenn Sie die Eindeutigkeit einer Spalte erzwingen müssen, die derzeit nicht als Primärschlüssel festgelegt ist oder keine eindeutige Einschränkung hat, wie z. B. E-Mail-Adressen in einer Benutzertabelle.

Die folgenden Beispiele zeigen, wie Sie eine Beispielbenutzertabelle erstellen, sie mit Testdaten füllen, die bekannte Duplikate enthalten, und dann die Erkennungsabfrage ausführen.

Definieren Sie das Tabellenschema

```
-- 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
);
```

Fügen Sie Beispieldaten ein, die Sätze doppelter E-Mail-Adressen enthalten

```
-- 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');
```

Führen Sie eine Abfrage zur Erkennung doppelter Objekte

```
-- 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;
```

Alle Datensätze mit doppelten E-Mail-Adressen anzeigen

```
 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)
```

Wenn wir die Anweisung zur Indexerstellung jetzt ausprobieren würden, würde sie fehlschlagen:

```
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)
```

Systemtabellen und Befehle in Aurora DSQL

In den folgenden Abschnitten erfahren Sie mehr über die unterstützten Systemtabellen und Kataloge in Aurora DSQL.

Systemtabellen

Aurora DSQL ist mit PostgreSQL kompatibel, sodass viele Systemkatalogtabellen und Ansichten von PostgreSQL auch in Aurora DSQL existieren.

Wichtige PostgreSQL-Katalogtabellen und -ansichten

In der folgenden Tabelle werden die gängigsten Tabellen und Ansichten beschrieben, die Sie in Aurora DSQL verwenden könnten.

Name	Beschreibung
`pg_namespace`	Informationen zu allen Schemas
`pg_tables`	Informationen zu allen Tabellen
`pg_attribute`	Informationen zu allen Attributen
`pg_views`	Informationen zu (vor-) definierten Ansichten
`pg_class`	Beschreibt alle Tabellen, Spalten, Indizes und ähnliche Objekte
`pg_stats`	Ein Blick auf die Statistiken des Planers
`pg_user`	Informationen über Benutzer
`pg_roles`	Informationen über Benutzer und Gruppen
`pg_indexes`	Listet alle Indizes auf
`pg_constraint`	Listet Einschränkungen für Tabellen auf

Unterstützte und nicht unterstützte Katalogtabellen

Die folgende Tabelle zeigt, welche Tabellen in Aurora DSQL unterstützt und welche nicht.

Name	Gilt für Aurora DSQL
`pg_aggregate`	Nein
`pg_am`	Ja
`pg_amop`	Nein

Name	Gilt für Aurora DSQL
pg_amproc	Nein
pg_attrdef	Ja
pg_attribute	Ja
pg_authid	Nein (verwendenpg_roles)
pg_auth_members	Ja
pg_cast	Ja
pg_class	Ja
pg_collation	Ja
pg_constraint	Ja
pg_conversion	Nein
pg_database	Nein
pg_db_role_setting	Ja
pg_default_acl	Ja
pg_depend	Ja
pg_description	Ja
pg_enum	Nein
pg_event_trigger	Nein
pg_extension	Nein
pg_foreign_data_wrapper	Nein
pg_foreign_server	Nein

Name	Gilt für Aurora DSQL
pg_foreign_table	Nein
pg_index	Ja
pg_inherits	Ja
pg_init_privs	Nein
pg_language	Nein
pg_largeobject	Nein
pg_largeobject_metadata	Ja
pg_namespace	Ja
pg_opclass	Nein
pg_operator	Ja
pg_opfamily	Nein
pg_parameter_acl	Ja
pg_partitioned_table	Nein
pg_policy	Nein
pg_proc	Nein
pg_publication	Nein
pg_publication_namespace	Nein
pg_publication_rel	Nein
pg_range	Ja
pg_replication_origin	Nein

Name	Gilt für Aurora DSQL
pg_rewrite	Nein
pg_seclabel	Nein
pg_sequence	Nein
pg_shdepend	Ja
pg_shdescription	Ja
pg_shseclabel	Nein
pg_statistic	Ja
pg_statistic_ext	Nein
pg_statistic_ext_data	Nein
pg_subscription	Nein
pg_subscription_rel	Nein
pg_tablespace	Nein
pg_transform	Nein
pg_trigger	Nein
pg_ts_config	Ja
pg_ts_config_map	Ja
pg_ts_dict	Ja
pg_ts_parser	Ja
pg_ts_template	Ja
pg_type	Ja

Name	Gilt für Aurora DSQL
pg_user_mapping	Nein

Unterstützte und nicht unterstützte Systemansichten

Die folgende Tabelle zeigt, welche Ansichten in Aurora DSQL unterstützt und welche nicht.

Name	Gilt für Aurora DSQL
pg_available_extensions	Nein
pg_available_extension_versions	Nein
pg_backend_memory_contexts	Ja
pg_config	Nein
pg_cursors	Nein
pg_file_settings	Nein
pg_group	Ja
pg_hba_file_rules	Nein
pg_ident_file_mappings	Nein
pg_indexes	Ja
pg_locks	Nein
pg_matviews	Nein
pg_policies	Nein
pg_prepared_statements	Nein
pg_prepared_xacts	Nein

Name	Gilt für Aurora DSQL
pg_publication_tables	Nein
pg_replication_origin_status	Nein
pg_replication_slots	Nein
pg_roles	Ja
pg_rules	Nein
pg_seclabels	Nein
pg_sequences	Nein
pg_settings	Ja
pg_shadow	Ja
pg_shmem_allocations	Ja
pg_stats	Ja
pg_stats_ext	Nein
pg_stats_ext_exprs	Nein
pg_tables	Ja
pg_timezone_abbrevs	Ja
pg_timezone_names	Ja
pg_user	Ja
pg_user_mappings	Nein
pg_views	Ja
pg_stat_activity	Nein

Name	Gilt für Aurora DSQL
pg_stat_replication	Nein
pg_stat_replication_slots	Nein
pg_stat_wal_receiver	Nein
pg_stat_recovery_prefetch	Nein
pg_stat_subscription	Nein
pg_stat_subscription_stats	Nein
pg_stat_ssl	Ja
pg_stat_gssapi	Nein
pg_stat_archiver	Nein
pg_stat_io	Nein
pg_stat_bgwriter	Nein
pg_stat_wal	Nein
pg_stat_database	Nein
pg_stat_database_conflicts	Nein
pg_stat_all_tables	Nein
pg_stat_all_indexes	Nein
pg_statio_all_tables	Nein
pg_statio_all_indexes	Nein
pg_statio_all_sequences	Nein
pg_stat_slru	Nein

Name	Gilt für Aurora DSQL
pg_statio_user_tables	Nein
pg_statio_user_sequences	Nein
pg_stat_user_functions	Nein
pg_stat_user_indexes	Nein
pg_stat_progress_analyze	Nein
pg_stat_progress_basebackup	Nein
pg_stat_progress_cluster	Nein
pg_stat_progress_create_index	Nein
pg_stat_progress_vacuum	Nein
pg_stat_sys_indexes	Nein
pg_stat_sys_tables	Nein
pg_stat_xact_all_tables	Nein
pg_stat_xact_sys_tables	Nein
pg_stat_xact_user_functions	Nein
pg_stat_xact_user_tables	Nein
pg_statio_sys_indexes	Nein
pg_statio_sys_sequences	Nein
pg_statio_sys_tables	Nein
pg_statio_user_indexes	Nein

Die Ansichten sys.jobs und sys.iam_pg_role_mappings

Aurora DSQL unterstützt die folgenden Systemansichten:

`sys.jobs`

`sys.jobs`bietet Statusinformationen über asynchrone Jobs. Nachdem Sie beispielsweise einen asynchronen Index erstellt haben, gibt Aurora DSQL a zurück. `job_uuid` Sie können dies `job_uuid` mit verwendensys.jobs, um den Status des Jobs nachzuschlagen.

```
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`

Die Ansicht `sys.iam_pg_role_mappings` enthält Informationen zu den Berechtigungen, die IAM-Benutzern gewährt wurden. Wenn `DQSLDBConnect` es sich beispielsweise um eine IAM-Rolle handelt, die Aurora DSQL-Zugriff für Nicht-Administratoren gewährt, und einem Benutzer mit dem Namen `testuser` die `DQSLDBConnect` Rolle und die entsprechenden Berechtigungen gewährt werden, können Sie die `sys.iam_pg_role_mappings` Ansicht abfragen, um zu sehen, welchen Benutzern welche Berechtigungen gewährt werden.

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

Die Tabelle pg_class

Die `pg_class` Tabelle speichert Metadaten zu Datenbankobjekten. Führen Sie den folgenden Befehl aus, um die ungefähre Anzahl der Zeilen in einer Tabelle zu ermitteln.

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

Daraufhin erhalten Sie ein Ergebnis, das dem hier dargestellten entspricht.

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

```
9.993836e+08
```

Der **ANALYZE** Befehl

Der ANALYZE Befehl sammelt Statistiken über den Inhalt von Tabellen in der Datenbank und speichert die Ergebnisse in der pg_stats Systemansicht. Anschließend verwendet der Abfrageplaner diese Statistiken, um die effizientesten Ausführungspläne für Abfragen zu ermitteln.

In Aurora DSQL können Sie den ANALYZE Befehl nicht innerhalb einer expliziten Transaktion ausführen. ANALYZEunterliegt nicht dem Timeout-Limit für Datenbanktransaktionen.

Um den Bedarf an manuellen Eingriffen zu reduzieren und die Statistiken stets auf dem neuesten Stand zu halten, wird Aurora DSQL automatisch ANALYZE als Hintergrundprozess ausgeführt. Dieser Hintergrundjob wird automatisch auf der Grundlage der beobachteten Änderungsrate in der Tabelle ausgelöst. Er ist mit der Anzahl der Zeilen (Tupel) verknüpft, die seit der letzten Analyse eingefügt, aktualisiert oder gelöscht wurden.

ANALYZEläuft asynchron im Hintergrund und seine Aktivität kann in der Systemansicht sys.jobs mit der folgenden Abfrage überwacht werden:

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

Die wichtigsten Überlegungen

> **Note**
>
> ANALYZEJobs werden wie andere asynchrone Jobs in Aurora DSQL abgerechnet. Wenn Sie eine Tabelle ändern, kann dies indirekt einen Auftrag zur automatischen Erfassung von Hintergrundstatistiken auslösen, was aufgrund der damit verbundenen Aktivität auf Systemebene zu Gebühren für die Datenerfassung führen kann.

Automatisch ausgelöste ANALYZE Hintergrundaufträge erfassen dieselben Arten von Statistiken wie bei manuellen Aufträgen ANALYZE und wenden sie standardmäßig auf Benutzertabellen an. System- und Katalogtabellen sind von diesem automatisierten Prozess ausgenommen.

Verwaltung von Aurora DSQL-Clustern

Aurora DSQL bietet mehrere Konfigurationsoptionen, mit denen Sie die richtige Datenbankinfrastruktur für Ihre Bedürfnisse einrichten können. Lesen Sie die folgenden Abschnitte, um Ihre Aurora DSQL-Cluster-Infrastruktur einzurichten.

Themen

Die in diesem Leitfaden erläuterten Merkmale und Funktionen stellen sicher, dass Ihre Aurora DSQL-Umgebung robuster und reaktionsschneller ist und Ihre Anwendungen bei deren Wachstum und Weiterentwicklung unterstützen kann.

Konfiguration von Clustern mit einer Region

Erstellen eines Clusters

Erstellen Sie einen Cluster mit dem create-cluster Befehl.

> **Note**
>
> Die Clustererstellung ist ein asynchroner Vorgang. Rufen Sie die GetCluster API auf, bis sich der Status auf `ACTIVE` ändert. Sie können eine Verbindung zu Ihrem Cluster herstellen, nachdem dieser aktiv ist.

Example Befehl

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

> **Note**
>
> Um den Löschschutz bei der Erstellung zu deaktivieren, fügen Sie das `--no-deletion-protection-enabled` Kennzeichen hinzu.

Example Antwort

```
{
    "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"
    }
}
```

Einen Cluster beschreiben

Rufen Sie mithilfe des get-cluster Befehls Informationen zu einem Cluster ab.

Example Befehl

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

Example Antwort

```
{
    "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"
    }
}
```

Aktualisieren eines Clusters

Aktualisieren Sie einen vorhandenen Cluster mithilfe des update-cluster Befehls.

> **Note**
>
> Updates sind asynchrone Vorgänge. Rufen Sie die GetCluster API auf, bis sich der Status ändert, `ACTIVE` um Ihre Änderungen zu sehen.

Example Befehl

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

Example Antwort

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

Löschen eines Clusters

Löschen Sie einen vorhandenen Cluster mit dem delete-cluster Befehl.

> **Note**
>
> Sie können nur Cluster löschen, bei denen der Löschschutz deaktiviert ist. Standardmäßig ist der Löschschutz aktiviert, wenn Sie neue Cluster erstellen.

Example Befehl

```
aws dsql delete-cluster \
```

```
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Antwort

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

Auflisten von Clustern

Listen Sie Ihre Cluster mit dem list-clusters Befehl auf.

Example Befehl

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

Example Antwort

```
{
    "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"
        }
    ]
}
```

Konfiguration von Clustern mit mehreren Regionen

In diesem Kapitel wird erklärt, wie mehrere AWS-Regionen Cluster konfiguriert und verwaltet werden.

Stellen Sie eine Verbindung zu Ihrem Cluster mit mehreren Regionen her

Peering-Cluster mit mehreren Regionen bieten zwei regionale Endpunkte, einen in jedem Peering-Cluster. AWS-Region Beide Endpunkte stellen eine einzige logische Datenbank dar, die gleichzeitige Lese- und Schreibvorgänge mit hoher Datenkonsistenz unterstützt. Zusätzlich zu Peering-Clustern verfügt ein Cluster mit mehreren Regionen auch über eine Zeugenregion, in der ein begrenztes Fenster verschlüsselter Transaktionsprotokolle gespeichert wird, was zur Verbesserung der Beständigkeit und Verfügbarkeit in mehreren Regionen verwendet wird. Zeugenregionen mit mehreren Regionen haben keine Endpunkte.

Cluster mit mehreren Regionen erstellen

Um Cluster mit mehreren Regionen zu erstellen, erstellen Sie zunächst einen Cluster mit einer Zeugenregion. Anschließend wird dieser Cluster mit einem zweiten Cluster verglichen, der dieselbe Zeugenregion wie Ihr erster Cluster teilt. Das folgende Beispiel zeigt, wie Cluster in USA Ost (Nord-Virginia) und USA Ost (Ohio) mit USA West (Oregon) als Zeugenregion erstellt werden.

Schritt 1: Erstellen Sie den ersten Cluster in USA Ost (Nord-Virginia)

Verwenden Sie den folgenden Befehl, um einen Cluster im Osten der USA (Nord-Virginia) AWS-Region mit Eigenschaften für mehrere Regionen zu erstellen.

```
aws dsql create-cluster \
--region us-east-1 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Antwort:

```
{
    "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**
>
> Wenn der API-Vorgang erfolgreich ist, wechselt der Cluster in den `PENDING_SETUP` Status. Die Clustererstellung bleibt so lange bestehen, `PENDING_SETUP` bis Sie den Cluster mit dem ARN seines Peer-Clusters aktualisieren.

Schritt 2: Erstellen Sie Cluster zwei in US East (Ohio)

Verwenden Sie den folgenden Befehl, um einen Cluster im Osten der USA (Ohio) AWS-Region mit Eigenschaften für mehrere Regionen zu erstellen.

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Antwort:

```
{
    "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"
        ]
    }
}
```

Wenn der API-Vorgang erfolgreich ist, wechselt der Cluster in den `PENDING_SETUP` Status. Die Clustererstellung bleibt so lange im `PENDING_SETUP` Status, bis Sie sie mit dem ARN eines anderen Clusters für das Peering aktualisieren.

Schritt 3: Peer-Cluster in USA Ost (Nord-Virginia) mit USA Ost (Ohio)

Verwenden Sie den `update-cluster` Befehl, um Ihr Cluster „USA Ost“ (Nord-Virginia) mit Ihrem Cluster „USA Ost“ (Ohio) als Peering zu verknüpfen. Geben Sie Ihren Clusternamen USA Ost (Nord-Virginia) und eine JSON-Zeichenfolge mit dem ARN des Clusters USA Ost (Ohio) an.

```
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 Antwort

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

Schritt 4: Peer-Cluster in USA Ost (Ohio) mit USA Ost (Nord-Virginia)

Verwenden Sie den `update-cluster` Befehl, um Ihren Cluster USA Ost (Ohio) mit Ihrem Cluster USA Ost (Nord-Virginia) als Peering zu verknüpfen. Geben Sie Ihren Clusternamen USA Ost (Ohio) und eine JSON-Zeichenfolge mit dem ARN des Clusters USA Ost (Nord-Virginia) an.

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 Antwort

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:51:16.145000-07:00"
}
```

> **Note**
>
> Nach erfolgreichem Peering wechseln beide Cluster vom Status „PENDING_SETUP“ in den Status „CREATING“ und schließlich in den Status „ACTIVE“, wenn sie einsatzbereit sind.

Eigenschaften von Clustern mit mehreren Regionen anzeigen

Wenn Sie einen Cluster beschreiben, können Sie sich die Eigenschaften mehrerer Regionen für Cluster in verschiedenen Regionen anzeigen lassen. AWS-Regionen

Example

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

Example Antwort

```
{
    "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"
       ]
    }
}
```

Peer-Cluster bei der Erstellung

Sie können die Anzahl der Schritte reduzieren, indem Sie bei der Clustererstellung Peering-Informationen einbeziehen. Nachdem Sie Ihren ersten Cluster in USA Ost (Nord-Virginia) erstellt

haben (Schritt 1), können Sie Ihren zweiten Cluster in USA Ost (Ohio) erstellen und gleichzeitig den Peering-Prozess einleiten, indem Sie den ARN des ersten Clusters einbeziehen.

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"]}'
```

Dies kombiniert die Schritte 2 und 4, aber Sie müssen immer noch Schritt 3 (Aktualisierung des ersten Clusters mit dem ARN des zweiten Clusters) abschließen, um die Peering-Beziehung herzustellen. Nachdem alle Schritte abgeschlossen sind, durchlaufen beide Cluster denselben Status wie im Standardprozess: von PENDING_SETUP zu CREATING und schließlich zu ACTIVE, wenn sie einsatzbereit sind.

Cluster mit mehreren Regionen löschen

Um einen Cluster mit mehreren Regionen zu löschen, müssen Sie zwei Schritte ausführen.

1. Schalten Sie den Löschschutz für jeden Cluster aus.
2. Löschen Sie jeden Peering-Cluster separat in seinem jeweiligen AWS-Region

Aktualisieren und löschen Sie den Cluster in USA Ost (Nord-Virginia)

1. Schalten Sie den Löschschutz mit dem `update-cluster` Befehl aus.

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

2. Löschen Sie den Cluster mit dem `delete-cluster` Befehl.

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

   Dieser Befehl gibt das folgende Antwort zurück.

```
{
    "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**
>
> Der Cluster wechselt in den `PENDING_DELETE` Status. Der Löschvorgang ist erst abgeschlossen, wenn Sie den Peering-Cluster in US East (Ohio) löschen.

Aktualisieren und löschen Sie den Cluster in USA Ost (Ohio)

1. Schalten Sie den Löschschutz mit dem `update-cluster` Befehl aus.

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

2. Löschen Sie den Cluster mit dem `delete-cluster` Befehl.

```
aws dsql delete-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5'
```

Der Befehl gibt die folgende Antwort zurück:

```
{
    "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**
>
> Der Cluster wechselt in den `PENDING_DELETE` Status. Nach einigen Sekunden versetzt das System nach der Validierung automatisch beide Peering-Cluster in den `DELETING` Status.

Konfiguration von Clustern mit mehreren Regionen mithilfe von AWS CloudFormation

Sie können dieselbe AWS CloudFormation Ressource verwenden, `AWS::DSQL::Cluster` um Aurora DSQL-Cluster mit einer oder mehreren Regionen bereitzustellen und zu verwalten.

Weitere Informationen zum Erstellen, Ändern und Verwalten von Clustern mithilfe der Ressource finden Sie in der Amazon Aurora `AWS::DSQL::Cluster` DSQL-Ressourcentyp-Referenz.

Erstellen der ersten Cluster-Konfiguration

Erstellen Sie zunächst eine AWS CloudFormation Vorlage zur Definition Ihres Clusters mit mehreren Regionen:

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

Erstellen Sie Stacks in beiden Regionen mit den folgenden AWS CLI-Befehlen:

```
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
```

Cluster-Identifikatoren finden

Rufen Sie die physische Ressource IDs für Ihre Cluster ab:

```
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"
]
```

Aktualisierung der Cluster-Konfiguration

Aktualisieren Sie Ihre AWS CloudFormation Vorlage so, dass sie beide Cluster enthält 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
```

Wenden Sie die aktualisierte Konfiguration auf beide Regionen an:

```
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
```

Programmieren mit Aurora DSQL

Aurora DSQL bietet Ihnen die folgenden Tools, um Ihre Aurora DSQL-Ressourcen programmgesteuert zu verwalten.

AWS Command Line Interface (AWS CLI)

Sie können Ihre Ressourcen mithilfe der Befehlszeilen-Shell erstellen und verwalten. AWS CLI Das AWS CLI bietet direkten Zugriff auf das APIs für AWS-Services, z. B. Aurora DSQL. Syntax und Beispiele für die Befehle für Aurora DSQL finden Sie unter dsql in der AWS CLI Befehlsreferenz.

AWS Kits für die Softwareentwicklung () SDKs

AWS bietet SDKs für viele beliebte Technologien und Programmiersprachen. Sie erleichtern es Ihnen, von Ihren Anwendungen AWS-Services aus in dieser Sprache oder Technologie anzurufen. Weitere Informationen zu diesen finden Sie SDKs unter Tools für die Entwicklung und Verwaltung von Anwendungen auf AWS.

Aurora DSQL-API

Diese API ist eine weitere Programmierschnittstelle für Aurora DSQL. Wenn Sie diese API verwenden, müssen Sie jede HTTPS-Anfrage korrekt formatieren und jeder Anfrage eine gültige digitale Signatur hinzufügen. Weitere Informationen finden Sie unter API-Referenz.

AWS CloudFormation

Das AWS::DSQL::Clusterist eine AWS CloudFormation Ressource, mit der Sie Aurora DSQL-Cluster als Teil Ihrer Infrastruktur als Code erstellen und verwalten können. AWS CloudFormation hilft Ihnen dabei, Ihre gesamte AWS Umgebung im Code zu definieren, was es einfacher macht, Ihre Infrastruktur konsistent und zuverlässig bereitzustellen, zu aktualisieren und zu replizieren.

Wenn Sie die AWS::DSQL::Cluster Ressource in Ihren AWS CloudFormation Vorlagen verwenden, können Sie Aurora DSQL-Cluster zusammen mit Ihren anderen Cloud-Ressourcen deklarativ bereitstellen. Dadurch wird sichergestellt, dass Ihre Dateninfrastruktur zusammen mit dem Rest Ihres Anwendungsstapels bereitgestellt und verwaltet wird.

Amazon Aurora DSQL SDKs, Treiber und Beispielcode

AWS Software Development Kits (SDKs) sind für viele gängige Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

Adapter und Dialekte

Die folgende Tabelle zeigt die verfügbaren ORM-Adapter und Datenbankdialekte für Aurora DSQL.

Programmiersprache	Framework	Link zum Repository
Java	Ruhezustand	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/

Codebeispiele

Clusterverwaltung mit SDK AWS

Die folgende Tabelle zeigt Codebeispiele für die Clusterverwaltung für verschiedene Programmiersprachen, die verwendet AWS SDKs werden.

Programmiersprache	Beispiel für einen Link zum Repository
C++	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/cluster_Verwaltung
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/dotnet/cluster
Go	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/go/cluster

Programmiersprache	Beispiel für einen Link zum Repository
Java	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/java/cluster
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/javascript/cluster
Python	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/python/cluster
Ruby	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/ruby/cluster
Rust	https://github.com/aws-samples/aurora-dsql-samples/_Verwaltung tree/main/rust/cluster

Beispiele für Treiber und objektrelationales Mapping () ORMs

Die folgende Tabelle zeigt Codebeispiele für Datenbanktreiber und ORM-Frameworks für verschiedene Programmiersprachen.

Programmiersprache	Treiber oder Framework	Beispiel für einen Link zum Repository
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

Programmiersprache	Treiber oder Framework	Beispiel für einen Link zum Repository
Java	Ruhezustand	https://github.com/awslabs/aurora-dsql-hibernate/-Klinik-App tree/main/examples/pet
Java	PGJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	Node-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	Psychokopie	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycho P 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-Klinik-App
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails

Programmiersprache	Treiber oder Framework	Beispiel für einen Link zum Repository
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx
TypeScript	Sequelisieren	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	Geben Sie ORM ein	https://github.com/aws-samples/aurora-dsql-samples/-orm tree/main/typescript/type

Aurora DSQL mit dem AWS CLI

In den folgenden Abschnitten erfahren Sie, wie Sie Ihre Cluster mit dem AWS CLI verwalten.

CreateCluster

Verwenden Sie den `create-cluster` Befehl, um einen Cluster zu erstellen.

> **Note**
>
> Die Clustererstellung erfolgt asynchron. Rufen Sie die `GetCluster` API auf, bis der Status lautet`ACTIVE`. Sie können eine Verbindung zu einem Cluster herstellen, sobald `ACTIVE` dieser

Beispiel für einen Befehl

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

> **Note**
>
> Wenn Sie den Löschschutz bei der Erstellung deaktivieren möchten, fügen Sie das `--no-deletion-protection-enabled` Kennzeichen hinzu.

Beispielantwort

```
{
    "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

Verwenden Sie den `get-cluster` Befehl, um einen Cluster zu beschreiben.

Beispiel für einen Befehl

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

Beispielantwort

```
{
    "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

Verwenden Sie den `update-cluster` Befehl, um einen vorhandenen Cluster zu aktualisieren.

> **Note**
>
> Aktualisierungen erfolgen asynchron. Rufen Sie die `GetCluster` API auf, bis der Status lautet `ACTIVE` und Sie werden die Änderungen beobachten.

Beispiel für einen Befehl

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

Beispielantwort

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

DeleteCluster

Verwenden Sie den `delete-cluster` Befehl, um einen vorhandenen Cluster zu löschen.

> **Note**
>
> Sie können nur einen Cluster löschen, für den der Löschschutz deaktiviert ist. Der Löschschutz ist standardmäßig aktiviert, wenn neue Cluster erstellt werden.

Beispiel für einen Befehl

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

Beispielantwort

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

ListClusters

Verwenden Sie den `list-clusters` Befehl, um das A von Clustern zu ermitteln.

Beispiel für einen Befehl

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

Beispielantwort

```
{
    "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 auf Clustern mit mehreren Regionen

Verwenden Sie den Befehl, um Informationen zu einem Cluster mit mehreren Regionen abzurufen. `get-cluster` Bei Clustern mit mehreren Regionen umfasst die Antwort den verknüpften Cluster. ARNs

Beispiel für einen Befehl

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

Beispielantwort

```
{
    "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"
    }
}
```

Aurora DSQL-Cluster erstellen, lesen, aktualisieren und löschen

CRUD-Beispiele (Create, Read, Update, Delete) stehen sowohl für Bereitstellungen mit einer Region als auch für Bereitstellungen mit mehreren Regionen zur Verfügung. Im Lieferumfang ist für jede Programmiersprache ein eigener `cluster_management` Abschnitt enthalten, in dem diese wichtigen Verwaltungsaufgaben veranschaulicht werden.

Bereitstellungen mit einer einzigen Region eignen sich ideal für Anwendungen, die Benutzer in einem bestimmten geografischen Gebiet bedienen. Sie bieten eine vereinfachte Verwaltung und eine geringere Latenz. Bereitstellungen in mehreren Regionen helfen Ihnen dabei, eine höhere Verfügbarkeit und Disaster Recovery-Funktionen zu erreichen, indem Sie Ihre Datenbank auf mehrere verteilen. AWS-Regionen

Wählen Sie den Bereitstellungstyp, der den Anforderungen Ihrer Anwendung in Bezug auf Verfügbarkeit, Leistung und geografische Verteilung entspricht.

Themen

Erstellen eines -Clusters

In den folgenden Informationen erfahren Sie, wie Sie Cluster mit einer oder mehreren Regionen in Aurora DSQL erstellen.

Python

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu erstellen.

```
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()
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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++

Im folgenden Beispiel können Sie einen Cluster in einem einzigen AWS-Region erstellen.

```
#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;
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
#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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu erstellen.

```
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();
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster in einem einzigen AWS-Region zu erstellen.

```
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);
        }
    }
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster in einem einzigen AWS-Region zu erstellen.

```
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(())
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster in einem einzigen AWS-Region zu erstellen.

```
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__
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster in einem einzigen AWS-Region zu erstellen.

```
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);
        }
    }
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster in einem einzigen AWS-Region zu erstellen.

```
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)
 }
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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)
 }

}
```

Holen Sie sich einen Cluster

In den folgenden Informationen erfahren Sie, wie Sie Informationen als Cluster in Aurora DSQL zurückgeben können.

Python

Verwenden Sie das folgende Beispiel, um Informationen über einen Cluster mit einer oder mehreren Regionen zu erhalten.

```
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++

Verwenden Sie das folgende Beispiel, um Informationen über einen Cluster mit einer oder mehreren Regionen abzurufen.

```
#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

Verwenden Sie das folgende Beispiel, um Informationen über einen Cluster mit einer oder mehreren Regionen abzurufen.

```
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

Mit dem folgenden Beispiel können Sie Informationen zu einem Cluster mit einer oder mehreren Regionen abrufen.

```
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

Im folgenden Beispiel können Sie Informationen zu einem Cluster mit einer oder mehreren Regionen abrufen.

```
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

Im folgenden Beispiel können Sie Informationen zu einem Cluster mit einer oder mehreren Regionen abrufen.

```
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

Im folgenden Beispiel können Sie Informationen zu einem Cluster mit einer oder mehreren Regionen abrufen.

```
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

Im folgenden Beispiel können Sie Informationen zu einem Cluster mit einer oder mehreren Regionen abrufen.

```
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)
 }
}
```

Aktualisieren eines -Clusters

In den folgenden Informationen erfahren Sie, wie Sie einen Cluster in Aurora DSQL aktualisieren. Die Aktualisierung eines Clusters kann ein oder zwei Minuten dauern. Wir empfehlen, einige Zeit zu warten und dann get cluster auszuführen, um den Status des Clusters abzurufen.

Python

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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++

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
#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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer oder mehreren Regionen zu aktualisieren.

```
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)
 }
}
```

Einen Cluster löschen

In den folgenden Informationen erfahren Sie, wie Sie einen Cluster in Aurora DSQL löschen.

Python

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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()
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen.

```
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++

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
#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;
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
#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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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();
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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);
        }
    }
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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(())
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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__
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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);
        }
    }
}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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

Verwenden Sie das folgende Beispiel AWS-Region, um einen Cluster in einem einzigen zu löschen.

```
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)
 }

}
```

Verwenden Sie das folgende Beispiel, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen eines Clusters mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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)
 }
}
```

Tutorials

Die folgenden Tutorials und der Beispielcode GitHub helfen Ihnen dabei, allgemeine Aufgaben in Aurora DSQL auszuführen.

- Verwendung von Benchbase mit Aurora DSQL — ein Zweig des Open-Source-Benchmarking-Dienstprogramms Benchbase, für den verifiziert wurde, dass er mit Aurora DSQL funktioniert.
- Aurora DSQL Loader — Dieses Open-Source-Python-Skript erleichtert Ihnen das Laden von Daten in Aurora DSQL für Ihre Anwendungsfälle, z. B. das Auffüllen von Tabellen zum Testen oder das Übertragen von Daten in Aurora DSQL.
- Aurora DSQL-Beispiele — `aws-samples/aurora-dsql-samples` Repository on GitHub enthält Codebeispiele für die Verbindung und Verwendung von Aurora DSQL in verschiedenen Programmiersprachen mithilfe von objektrelationalen Mappern () ORMs und Web-Frameworks. AWS SDKs Die Beispiele zeigen, wie allgemeine Aufgaben wie die Installation von Clients, die Authentifizierung und die Durchführung von CRUD-Vorgängen ausgeführt werden.

Verwendung AWS Lambda mit Amazon Aurora DSQL

Das folgende Tutorial beschreibt, wie Lambda mit Aurora DSQL verwendet wird.

Voraussetzungen

- Autorisierung zur Erstellung von Lambda-Funktionen. Weitere Informationen finden Sie unter Erste Schritte mit Lambda.

- Autorisierung zum Erstellen oder Ändern der von Lambda erstellten IAM-Richtlinie. Sie benötigen zwei Berechtigungen `iam:CreatePolicy` und. `iam:AttachRolePolicy` Weitere Informationen finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für IAM.
- Sie müssen npm v8.5.3 oder höher installiert haben.
- Sie müssen Zip v3.0 oder höher installiert haben.

Erstellen Sie eine neue Funktion in AWS Lambda.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AWS Lambda Konsole unter https://console.aws.amazon.com/lambda/.
2. Wählen Sie Funktion erstellen.
3. Geben Sie einen Namen ein, z. `dsql-sample` B.
4. Bearbeiten Sie die Standardeinstellungen nicht, um sicherzustellen, dass Lambda eine neue Rolle mit grundlegenden Lambda-Berechtigungen erstellt.
5. Wählen Sie Funktion erstellen.

Autorisieren Sie Ihre Lambda-Ausführungsrolle, um eine Verbindung zu Ihrem Cluster herzustellen

1. Wählen Sie in Ihrer Lambda-Funktion Konfiguration > Berechtigungen.
2. Wählen Sie den Rollennamen, um die Ausführungsrolle in der IAM-Konsole zu öffnen.
3. Wählen Sie „Berechtigungen hinzufügen“ > „Inline-Richtlinie erstellen“ und verwenden Sie den JSON-Editor.
4. Fügen Sie in Aktion die folgende Aktion ein, um Ihre IAM-Identität zu autorisieren, mithilfe der Admin-Datenbankrolle eine Verbindung herzustellen.

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

> **Note**
>
> Wir verwenden eine Administratorrolle, um die Anzahl der erforderlichen Schritte für den Einstieg zu minimieren. Sie sollten keine Administrator-Datenbankrolle für Ihre Produktionsanwendungen verwenden. Unter erfahren Verwenden von Datenbankrollen und IAM-Authentifizierung Sie, wie Sie benutzerdefinierte Datenbankrollen mit der

Autorisierung erstellen, die über die wenigsten Berechtigungen für Ihre Datenbank verfügt.

5. Fügen Sie unter Ressource den Amazon-Ressourcennamen (ARN) Ihres Clusters hinzu. Sie können auch einen Platzhalter verwenden.

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

6. Wählen Sie Weiter aus.
7. Geben Sie einen Namen für die Richtlinie ein, z. B. `dsql-sample-dbconnect`
8. Wählen Sie Richtlinie erstellen aus.

Erstellen Sie ein Paket, das auf Lambda hochgeladen werden soll.

1. Erstellen Sie einen Ordner mit dem Namen`myfunction`.
2. Erstellen Sie in dem Ordner eine neue Datei `package.json` mit dem folgenden Inhalt.

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

3. Erstellen Sie in dem Ordner eine Datei mit dem Namen `index.mjs` in dem Verzeichnis mit dem folgenden Inhalt.

```
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. Verwenden Sie die folgenden Befehle, um ein Paket zu erstellen.

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

Laden Sie das Codepaket hoch und testen Sie Ihre Lambda-Funktion

1. Wählen Sie auf der Registerkarte Code Ihrer Lambda-Funktion die Option Upload from > .zip-Datei
2. Laden Sie die von `pkg.zip` Ihnen erstellte Datei hoch. Weitere Informationen finden Sie unter Bereitstellen von Lambda-Funktionen von Node.js mit ZIP-Dateiarchiven.
3. Fügen Sie auf der Registerkarte Test Ihrer Lambda-Funktion die folgende JSON-Payload ein und ändern Sie sie so, dass sie Ihre Cluster-ID verwendet.
4. Verwenden Sie auf der Registerkarte Test Ihrer Lambda-Funktion den folgenden Event-JSON, der geändert wurde, um den Endpunkt Ihres Clusters anzugeben.

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

5. Geben Sie einen Namen für das Ereignis ein, z. B. `dsql-sample-test` Wählen Sie Speichern.
6. Wählen Sie Test aus.
7. Wählen Sie Details, um die Ausführungsantwort und die Protokollausgabe zu erweitern.
8. Wenn dies erfolgreich war, sollte die Antwort auf die Ausführung der Lambda-Funktion einen Statuscode 200 zurückgeben.

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

   Wenn die Datenbank einen Fehler zurückgibt oder wenn die Verbindung zur Datenbank fehlschlägt, gibt die Antwort auf die Ausführung der Lambda-Funktion den Statuscode 500 zurück.

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

Backup und Wiederherstellung für Amazon Aurora DSQL

Amazon Aurora DSQL hilft Ihnen dabei AWS Backup, Ihre Anforderungen an die Einhaltung gesetzlicher Vorschriften und die Geschäftskontinuität durch die Integration mit einem vollständig verwalteten Datenschutzservice zu erfüllen, der die Zentralisierung und Automatisierung von Backups zwischen AWS Services, in der Cloud und vor Ort erleichtert. Der Service optimiert die Erstellung, Verwaltung und Wiederherstellung von Backups für Aurora DSQL-Cluster mit einer oder mehreren Regionen.

Nachstehend sind einige der wichtigsten Funktionen aufgelistet:

- Zentralisiertes Backup-Management über das SDK oder AWS Management Console AWS CLI
- Vollständige Cluster-Backups
- Automatisierte Backup-Zeitpläne und Aufbewahrungsrichtlinien
- Regions- und kontoübergreifende Funktionen
- WORM-Konfiguration (Write-Once, Read-Many) für alle Backups, die Sie speichern

Weitere Informationen zu den Funktionen von AWS Backup Vault Lock und eine umfangreiche Liste der verfügbaren AWS Backup Funktionen für Aurora DSQL finden Sie unter Vorteile und Verfügbarkeit von AWS Backup Funktionen von Vault Lock im AWS Backup Entwicklerhandbuch.

Erste Schritte mit AWS Backup

AWS Backup erstellt vollständige Kopien Ihrer Aurora DSQL-Cluster. Sie können mit der Verwendung AWS Backup von Aurora DSQL beginnen, indem Sie die Schritte unter Erste Schritte mit AWS Backup befolgen:

1. Erstellen Sie On-Demand-Backups für sofortigen Schutz.
2. Erstellen Sie Backup-Pläne für automatisierte, geplante Backups.
3. Konfigurieren Sie Aufbewahrungsfristen und regionsübergreifendes Kopieren.
4. Richten Sie die Überwachung und Benachrichtigungen für Backup-Aktivitäten ein.

Wiederherstellung Ihrer Backups

Wenn Sie Aurora DSQL-Cluster wiederherstellen, werden AWS Backup immer neue Cluster erstellt, um Ihre Quelldaten zu erhalten.

Wiederherstellung von Clustern mit einer Region

Um einen Aurora DSQL Single-Region-Cluster wiederherzustellen, verwenden Sie die Konsole: https://console.aws.amazon.com/backup oder CLI, um den Wiederherstellungspunkt (Backup) auszuwählen, den Sie wiederherstellen möchten. Konfigurieren Sie die Einstellungen für den neuen Cluster, der aus Ihrem Backup erstellt wird. Ausführliche Anweisungen finden Sie unter Wiederherstellen eines Aurora DSQL-Clusters mit einer Region.

Cluster mit mehreren Regionen wiederherstellen

Die Wiederherstellung eines Aurora DSQL-Clusters mit mehreren Regionen wird sowohl über die Konsole: https://console.aws.amazon.com/backup als auch über die unterstützt. AWS CLI Eine ausführliche Anleitung finden Sie unter Einen Aurora DSQL-Cluster mit mehreren Regionen wiederherstellen.

Für die Wiederherstellung auf einem Aurora DSQL-Cluster mit mehreren Regionen können Sie ein Backup verwenden, das in einem einzigen erstellt wurde. AWS-Region Bevor Sie den Wiederherstellungsvorgang einleiten, müssen Sie jedoch sicherstellen, dass AWS-Regionen für Ihre Cluster mit mehreren Regionen insgesamt eine identische Kopie Ihres Backups vorhanden ist. Wenn Sie noch nicht über diese Kopien verfügen, müssen Sie das Backup zunächst auf ein anderes kopieren AWS-Region , das Cluster mit mehreren Regionen unterstützt.

Wir empfehlen, Sicherungskopien in Key AWS-Regionen zu erstellen, um robuste Notfallwiederherstellungsoptionen zu ermöglichen und Compliance-Anforderungen zu erfüllen. Informationen zur Ansicht, AWS-Regionen die für Aurora DSQL verfügbar sind, finden Sie unterthe section called “AWS-Region Verfügbarkeit”.

Detaillierte Anweisungen zu diesen Schritten finden Sie in der Amazon Aurora DSQL-Wiederherstellungsdokumentation.

Überwachung und Einhaltung der Vorschriften

AWS Backup bietet umfassende Einblicke in Sicherungs- und Wiederherstellungsvorgänge mit den folgenden Ressourcen.

- Ein zentrales Dashboard zur Nachverfolgung von Sicherungs- und Wiederherstellungsaufträgen
- Integration mit CloudWatch und CloudTrail.
- AWS Backup Audit Manager für Compliance-Berichterstattung und Prüfung.

Weitere Informationen Protokollierung von Aurora DSQL-Vorgängen mit AWS CloudTrail zum Protokollieren von Aufzeichnungen über Aktionen, die ein Benutzer, eine Rolle oder eine AWS-Service Zeit lang mit Aurora DSQL ausgeführt hat, finden Sie unter.

Weitere Ressourcen

Weitere Informationen zu AWS Backup Funktionen und deren Verwendung zusammen mit Aurora DSQL finden Sie in den folgenden Ressourcen:

- Verwaltete Richtlinien für AWS Backup
- Amazon Aurora DSQL-Wiederherstellung
- Unterstützte Dienste von AWS-Region
- Verschlüsselung für Backups in AWS Backup

Durch AWS Backup die Verwendung von Aurora DSQL implementieren Sie eine robuste, konforme und automatisierte Backup-Strategie, die Ihre kritischen Datenbankressourcen schützt und gleichzeitig den Verwaltungsaufwand minimiert. Egal, ob Sie einen einzelnen Cluster oder eine komplexe Bereitstellung in mehreren Regionen verwalten, AWS Backup bietet die Tools, die Sie benötigen, um sicherzustellen, dass Ihre Daten sicher und wiederherstellbar bleiben.

Überwachung und Protokollierung für Aurora DSQL

Überwachung und Protokollierung sind ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit und Leistung Ihrer Amazon Aurora DSQL-Ressourcen. Sie sollten Protokolldaten aus allen Teilen Ihrer Aurora DSQL-Ressourcen überwachen und sammeln, damit Sie einen Fehler an mehreren Punkten problemlos debuggen können.

- Amazon CloudWatch überwacht Ihre AWS Ressourcen und die Anwendungen, auf denen Sie laufen, AWS in Echtzeit. Sie können Kennzahlen erfassen und verfolgen, benutzerdefinierte Dashboards erstellen und Alarme festlegen, die Sie benachrichtigen oder Maßnahmen ergreifen, wenn eine bestimmte Metrik einen von Ihnen festgelegten Schwellenwert erreicht. Sie können beispielsweise die CPU-Auslastung oder andere Kennzahlen Ihrer EC2 Amazon-Instances CloudWatch verfolgen und bei Bedarf automatisch neue Instances starten. Weitere Informationen finden Sie im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrailerfasst API-Aufrufe und zugehörige Ereignisse, die von Ihnen oder in Ihrem Namen getätigt wurden, AWS-Konto und übermittelt die Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket. Sie können feststellen, welche Benutzer und Konten angerufen wurden AWS, von welcher Quell-IP-Adresse aus die Anrufe getätigt wurden und wann die Aufrufe erfolgten. Weitere Informationen finden Sie im AWS CloudTrail -Benutzerhandbuch.

Aurora DSQL-Clusterstatus anzeigen

Der Aurora DSQL-Clusterstatus liefert wichtige Informationen über den Zustand und die Konnektivität des Clusters. Sie können den Status von Clustern und Cluster-Instances mithilfe der AWS Management Console Aurora DSQL-API anzeigen. AWS CLI

Status und Definitionen des Aurora DSQL-Clusters

Die folgende Tabelle beschreibt jeden möglichen Status für einen Aurora DSQL-Cluster und was jeder Status bedeutet.

Status	Description
Erstellen	Aurora DSQL versucht, Ressourcen für den Cluster zu erstellen oder zu konfigurieren. Alle Verbindungsversuche schlagen fehl, solange sich ein Cluster in diesem Status befindet.

Status	Description
Aktiv	Der Cluster ist betriebsbereit und einsatzbereit.
Inaktiv	Ein Cluster wird inaktiv, wenn er lange genug inaktiv ist, damit Aurora DSQL die für ihn konfigurierten Ressourcen zurückfordern kann. Wenn Sie eine Verbindung zu einem Cluster im Leerlauf herstellen, versetzt Aurora DSQL den Cluster wieder in den Status Aktiv.
Inaktiv	Ein Cluster wird inaktiv, wenn auf dem Cluster über einen längeren Zeitraum keine Aktivität stattgefunden hat. Wenn Sie versuchen, eine Verbindung zu einem inaktiven Cluster herzustellen, versetzt Aurora DSQL den Cluster automatisch wieder in den Status Aktiv.
Wird aktualisiert	Ein Cluster wechselt in den Status Aktualisierung, wenn Sie Änderungen an der Cluster-Konfiguration vornehmen.
Wird gelöscht	Ein Cluster wechselt in den Status Löschen, wenn Sie eine Anfrage zum Löschen einreichen.
Deleted (Gelöscht)	Der Cluster wurde erfolgreich gelöscht.
Fehlgeschlagen	Aurora DSQL konnte den Cluster nicht erstellen, da ein Fehler aufgetreten ist.
Das Setup steht noch aus	Nur für Cluster mit mehreren Regionen. Ein Cluster mit mehreren Regionen erhält den Status Ausstehende Einrichtung, wenn Sie in Ihrer ersten Region einen Cluster mit mehreren Regionen mit einer Zeugenregion erstellen. Die Clustererstellung wird angehalten, bis Sie einen weiteren Cluster in einer sekundären Region erstellen und die beiden Cluster miteinander verbinden.
Ausstehender Löschvorgang	Nur für Cluster mit mehreren Regionen. Ein Cluster mit mehreren Regionen erhält den Status „Ausstehender Löschvorgang“, wenn Sie einen Cluster aus dem Cluster löschen. Der Cluster wechselt in den Status Löschen, sobald Sie den letzten Peer-Cluster gelöscht haben.

Ihren Aurora DSQL-Clusterstatus anzeigen

Um den Status Ihres Clusters anzuzeigen, verwenden Sie die AWS Management Console AWS CLI, oder Aurora DSQL API.

Konsole

Gehen Sie wie folgt vor, um den Cluster-Status in folgender AWS Management Console Datei anzuzeigen:

So zeigen Sie den Clusterstatus in der Konsole an

1. Öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Wählen Sie im Navigationsbereich Clusters (Cluster) aus.
3. Sehen Sie sich den Status für jeden Cluster im Dashboard an.

AWS CLI

Verwenden Sie den folgenden AWS CLI Befehl, um den Status eines einzelnen Clusters zu überprüfen.

```
aws dsql get-cluster --identifier cluster-id --query status --output text
```

Führen Sie den folgenden Befehl aus, um den Status aller Cluster aufzulisten.

```
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
```

Diese Beispielausgabe zeigt zwei aktive Cluster und einen Cluster, der gerade gelöscht wird.

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

Überwachung von Aurora DSQL mit Amazon CloudWatch

Überwachen Sie mithilfe von Aurora DSQL CloudWatch, das Rohdaten sammelt und zu lesbaren Metriken verarbeitet, die nahezu in Echtzeit verfügbar sind. CloudWatch speichert diese Statistiken 15 Monate lang, sodass Sie einen besseren Überblick über die Leistung Ihrer Webanwendung oder Ihres Dienstes erhalten. Stellen Sie Alarme ein, um auf bestimmte Schwellenwerte zu achten, und senden Sie Benachrichtigungen oder ergreifen Sie Maßnahmen, wenn sie erreicht werden. Sehen Sie sich die folgenden Nutzungs- und Beobachtbarkeitsmetriken an, die für Aurora DSQL verfügbar sind.

Weitere Informationen finden Sie im CloudWatch Amazon-Benutzerhandbuch.

Beobachtbarkeit und Leistung

In dieser Tabelle werden Observabilitätsmetriken für Aurora DSQL beschrieben. Sie enthält Metriken für die Nachverfolgung schreibgeschützter Transaktionen und der Gesamtzahl der Transaktionen, um eine allgemeine Beschreibung der Arbeitslast zu ermöglichen. Umsetzbare Messwerte wie Abfrage-Timeouts und OCC-Konfliktrate sind enthalten, um Leistungsprobleme und Parallelitätskonflikte zu identifizieren. Sitzungsbezogene Metriken, sowohl aktive als auch Gesamtwerte, bieten Einblicke in die aktuelle Systemauslastung.

CloudWatch Metrikname	Metrik	Einheit	Beschreibung
ReadOnlyTransactions	Schreibgeschützte Transaktionen	Keine	Die Anzahl der schreibgeschützten Transaktionen
TotalTransactions	Transaktionen insgesamt	Keine	Die Gesamtzahl der auf dem System ausgeführten Transaktionen, einschließlich schreibgeschützter Transaktionen.
QueryTimeouts	Zeitüberschreitungen bei Abfragen	Keine	Die Anzahl der Abfragen, bei denen

CloudWatch Metrikname	Metrik	Einheit	Beschreibung
			das Zeitlimit überschri tten wurde, weil die maximale Transakti onszeit erreicht wurde
OccConflicts	OCC-Konflikte	Keine	Die Anzahl der Transaktionen, die aufgrund von OCC auf Schlüsselebene abgebrochen wurden
CommitLatency	Latenz festschreiben	Millisekunden	Zeitaufwand für die Commit-Phase der Abfrageausführung (P50)
BytesWritten	Geschriebene Byte	bytes	In den Speicher geschriebene Bytes
BytesRead	Gelesene Bytes	bytes	Aus dem Speicher gelesene Bytes
ComputeTime	QP-Rechenzeit	Millisekunden	QP-Wanduhrzeit
ClusterStorageSize	Größe des Cluster-S peichers	bytes	Cluster-Größe

Nutzungsmetriken

Aurora DSQL misst alle anforderungsbasierten Aktivitäten, wie Abfrageverarbeitung, Lese- und Schreibvorgänge, mithilfe einer einzigen normalisierten Abrechnungseinheit, der Distributed Processing Unit (DPU).

CloudWatch Metrikname	Metrik	Dimension: ResourceId	Einheit	Beschreibung
Geschriebene CPU	Einheiten schreiben	<cluster-id>	DPU	Nähert sich der aktiven Schreibkomponente Ihrer Aurora DSQL-Cluster-DPU-Nutzung.
MultiRegionWriteDPU	Schreibeinheiten für mehrere Regionen	<cluster-id>	DPU	Anwendbar für Multi-Region-Cluster: Nähert sich der Multi-Region-Write-Active-Use-Komponente Ihrer Aurora DSQL-Cluster-DPU-Nutzung an.
DPU lesen	Einheiten lesen	<cluster-id>	DPU	Nähert sich der aktiven Lesekomponente Ihrer Aurora DSQL-Cluster-DPU-Nutzung.
Berechnete CPU	Einheiten berechnen	<cluster-id>	DPU	Nähert sich der Rechenkomponente für aktive Nutzung der DPU-Nutzung Ihres Aurora DSQL-Clusters.

CloudWatch Metrikname	Metrik	Dimension: ResourceId	Einheit	Beschreibung
Gesamt-DPU	Einheiten insgesamt	<cluster-id>	DPU	Nähert sich der gesamten aktiven Nutzungsk omponente Ihrer Aurora DSQL-Cluster-DPU-N utzung.

Protokollierung von Aurora DSQL-Vorgängen mit AWS CloudTrail

Amazon Aurora DSQL ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem AWS-Service ausgeführten Aktionen bereitstellt. Es gibt zwei Arten von Ereignissen CloudTrail: Verwaltungsereignisse und Datenereignisse. Verwaltungsereignisse werden ausgelöst, um Änderungen der AWS Ressourcenkonfiguration zu überprüfen. Datenereignisse erfassen die AWS Ressourcennutzung in der Regel auf der Servicedatenebene.

CloudTrail erfasst alle API-Aufrufe für Aurora DSQL als Ereignisse. Aurora DSQL zeichnet Konsolenaktivitäten als Managementereignisse auf. Es erfasst auch authentifizierte Verbindungsversuche zu Clustern als Datenereignisse.

Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Aurora DSQL gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, den Zeitpunkt der Anfrage, die Benutzeridentität, die die Anfrage gestellt hat, und weitere Details ermitteln.

CloudTrail ist standardmäßig aktiviert, AWS-Konto wenn Sie das Konto erstellen und Sie Zugriff auf den CloudTrail Eventverlauf haben. Der CloudTrail Ereignisverlauf bietet eine einsehbare, durchsuchbare, herunterladbare und unveränderliche Aufzeichnung der aufgezeichneten Verwaltungsereignisse der letzten 90 Tage in einem. AWS-Region Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter Arbeiten mit dem CloudTrail Ereignisverlauf. Für die Aufzeichnung des Ereignisverlaufs CloudTrail fallen keine Gebühren an.

Um eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto zu erstellen, einschließlich Ereignissen für Aurora DSQL, erstellen Sie einen Trail- oder AWS CloudTrail Lake-

Ereignisdatenspeicher (eine zentrale Speicher- und Analyselösung für AWS CloudTrail Ereignisse). Weitere Informationen zum Erstellen von Pfaden finden Sie unter Mit CloudTrail Pfaden arbeiten. Informationen zum Einrichten und Verwalten von Ereignisdatenspeichern finden Sie unter CloudTrail Lake Event Data Stores.

Aurora DSQL-Managementereignisse in CloudTrail

CloudTrail Verwaltungsereignisse bieten Informationen über Verwaltungsvorgänge, die mit Ressourcen in Ihrem AWS Konto ausgeführt werden. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. CloudTrail Erfasst standardmäßig Verwaltungsereignisse im Ereignisverlauf.

Amazon Aurora DSQL protokolliert alle Operationen der Aurora DSQL-Steuerebene als Managementereignisse. Eine Liste der Amazon Aurora DSQL-Steuerebenenoperationen, bei denen Aurora DSQL protokolliert CloudTrail, finden Sie in der Aurora DSQL API-Referenz.

Protokolle der Kontrollebene

Amazon Aurora DSQL protokolliert die folgenden Operationen der Aurora DSQL-Steuerebene CloudTrail als Managementereignisse.

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

Protokolle Backup und wiederherstellen

Amazon Aurora DSQL protokolliert die folgenden Aurora DSQL-Backup- und Wiederherstellungsvorgänge CloudTrail als Verwaltungsereignisse.

- `StartBackupJob`

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

Weitere Informationen zum Schutz Ihrer Aurora DSQL-Cluster mithilfe von finden Sie AWS Backup unterBackup und Wiederherstellung für Amazon Aurora DSQL.

AWS KMS-Protokolle

Amazon Aurora DSQL protokolliert die folgenden AWS KMS Vorgänge CloudTrail als Verwaltungsereignisse.

- `GenerateDataKey`
- `Decrypt`

Weitere Informationen darüber, wie CloudTrail Logs Anfragen verfolgen, an die Aurora DSQL in AWS KMS Ihrem Namen sendet, finden Sie unterÜberwachung der Aurora DSQL-Interaktion mit AWS KMS.

Aurora DSQL-Datenereignisse in CloudTrail

CloudTrail Datenereignisse liefern in der Regel Informationen über die Ressourcenoperationen, die auf oder in einer Ressource ausgeführt werden. Diese werden auch verwendet, um die Datenebenenoperationen des Dienstes zu erfassen. Datenereignisse sind oft Aktivitäten mit hohem Volume. Protokolliert standardmäßig CloudTrail keine Datenereignisse. Der CloudTrail Ereignisverlauf zeichnet keine Datenereignisse auf.

Weitere Informationen zum Protokollieren von Datenereignissen finden Sie unter Protokollieren von Datenereignissen mit dem AWS Management Console und Protokollieren von Datenereignissen mit dem AWS Command Line Interface im AWS CloudTrail -Benutzerhandbuch.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter AWS CloudTrail Preisgestaltung.

CloudTrail Erfasst für Aurora DSQL jeden Verbindungsversuch mit einem Aurora DSQL-Cluster als Datenereignis. In der folgenden Tabelle sind die Aurora DSQL-Ressourcentypen aufgeführt,

für die Sie Datenereignisse protokollieren können. In der Spalte Ressourcentyp (Konsole) wird der Wert angezeigt, der aus der Liste Ressourcentyp auf der CloudTrail Konsole ausgewählt werden kann. In der Wertspalte resources.type wird der `resources.type` Wert angezeigt, den Sie bei der Konfiguration erweiterter Event-Selektoren mithilfe von oder angeben würden. AWS CLI CloudTrail APIs In der CloudTrail Spalte APIs Protokollierte Daten werden die API-Aufrufe angezeigt, die CloudTrail für den Ressourcentyp protokolliert wurden.

Ressourcentyp (Konsole)	resources.type-Wert	Daten APIs wurden protokolliert CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Sie können erweiterte Event-Selektoren so konfigurieren, dass sie nach den `resources.ARN` Feldern `eventName` und filtern, sodass nur gefilterte Ereignisse protokolliert werden. Weitere Informationen zu diesen Kontingenten finden Sie unter AdvancedFieldSelector in derAWS CloudTrail -API-Referenz.

Das folgende Beispiel zeigt, wie die Konfiguration `dsql-data-events-trail` für AWS CLI den Empfang von Datenereignissen für Aurora DSQL konfiguriert wird.

```
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"] } ]}]'
```

Sicherheit in Amazon Aurora DSQL

Cloud-Sicherheit hat AWS höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für Amazon Aurora DSQL gelten, finden Sie unter AWS Services in Scope by Compliance Program AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Service, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von Aurora DSQL anwenden können. In den folgenden Themen erfahren Sie, wie Sie Aurora DSQL konfigurieren, um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie lernen auch, wie Sie andere AWS Dienste nutzen können, die Ihnen helfen, Ihre Aurora DSQL-Ressourcen zu überwachen und zu sichern.

Themen

- Infrastruktursicherheit in Amazon Aurora DSQL
- Konfiguration und Schwachstellenanalyse in Amazon Aurora DSQL
- Serviceübergreifende Confused-Deputy-Prävention
- Bewährte Sicherheitsmethoden für Aurora DSQL

AWS verwaltete Richtlinien für Amazon Aurora DSQL

Eine AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von erstellt und verwaltet AWS wird. AWS Verwaltete Richtlinien dienen dazu, Berechtigungen für viele gängige Anwendungsfälle bereitzustellen, sodass Sie damit beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS verwaltete Richtlinien für Ihre speziellen Anwendungsfälle möglicherweise keine Berechtigungen mit den geringsten Rechten gewähren, da sie allen AWS Kunden zur Verfügung stehen. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Sie können die in AWS verwalteten Richtlinien definierten Berechtigungen nicht ändern. Wenn die in einer AWS verwalteten Richtlinie definierten Berechtigungen AWS aktualisiert werden, wirkt sich das Update auf alle Prinzidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert eine AWS verwaltete Richtlinie höchstwahrscheinlich, wenn eine neue Richtlinie eingeführt AWS-Service wird oder neue API-Operationen für bestehende Dienste verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

AWS verwaltete Richtlinie: AmazonAurora DSQLFull Zugriff

Sie können Verbindungen `AmazonAuroraDSQLFullAccess` zu Ihren Benutzern, Gruppen und Rollen herstellen.

Diese Richtlinie gewährt Berechtigungen, die vollen Administratorzugriff auf Aurora DSQL ermöglichen. Principals mit diesen Berechtigungen können Aurora DSQL-Cluster, einschließlich Clustern mit mehreren Regionen, erstellen, löschen und aktualisieren. Sie können Tags zu Clustern hinzufügen und daraus entfernen. Sie können Cluster auflisten und Informationen zu einzelnen

Clustern anzeigen. Sie können Tags sehen, die an Aurora DSQL-Cluster angehängt sind. Sie können sich wie jeder andere Benutzer, auch als Administrator, mit der Datenbank verbinden. Sie können Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster ausführen, einschließlich des Startens, Stoppens und Überwachen von Sicherungs- und Wiederherstellungsaufträgen.
Die Richtlinie umfasst AWS KMS Berechtigungen, die Operationen mit vom Kunden verwalteten Schlüsseln ermöglichen, die für die Cluster-Verschlüsselung verwendet werden. Sie können alle Metriken in CloudWatch Ihrem Konto einsehen. Sie sind auch berechtigt, dienstbezogene Rollen für den `dsql.amazonaws.com` Service zu erstellen, was für die Erstellung von Clustern erforderlich ist.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt Principals vollen Zugriff auf Aurora DSQL.
- `cloudwatch`— erteilt die Erlaubnis, metrische Datenpunkte auf Amazon CloudWatch zu veröffentlichen.
- `iam`— erteilt die Berechtigung zum Erstellen einer dienstbezogenen Rolle.
- `backup and restore`— gewährt Berechtigungen zum Starten, Stoppen und Überwachen von Sicherungs- und Wiederherstellungsaufträgen für Aurora DSQL-Cluster.
- `kms`— gewährt die erforderlichen Berechtigungen, um den Zugriff auf vom Kunden verwaltete Schlüssel zu validieren, die für die Aurora DSQL-Clusterverschlüsselung beim Erstellen, Aktualisieren oder Herstellen einer Verbindung zu Clustern verwendet werden.

Sie finden die `AmazonAuroraDSQLFullAccess` Richtlinie auf der IAM-Konsole und AmazonAuroraDSQLFullAccess im Referenzhandbuch für AWS verwaltete Richtlinien.

AWS verwaltete Richtlinie: AmazonAurora DSQLRead OnlyAccess

Sie können `AmazonAuroraDSQLReadOnlyAccess` sie Ihren Benutzern, Gruppen und Rollen zuordnen.

Ermöglicht den Lesezugriff auf Aurora DSQL. Principals mit diesen Berechtigungen können Cluster auflisten und Informationen zu einzelnen Clustern einsehen. Sie können die Tags sehen, die an Aurora DSQL-Cluster angehängt sind. Sie können alle Metriken aus CloudWatch Ihrem Konto abrufen und einsehen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt allen Ressourcen in Aurora DSQL nur Leseberechtigungen.
- `cloudwatch`— erteilt die Erlaubnis, Batchmengen von CloudWatch metrischen Daten abzurufen und metrische Berechnungen mit abgerufenen Daten durchzuführen

Sie finden die `AmazonAuroraDSQLReadOnlyAccess` Richtlinie auf der IAM-Konsole und AmazonAuroraDSQLReadOnlyAccessim AWS Managed Policy Reference Guide.

AWS verwaltete Richtlinie: AmazonAurora DSQLConsole FullAccess

Sie können `AmazonAuroraDSQLConsoleFullAccess` sie Ihren Benutzern, Gruppen und Rollen zuordnen.

Ermöglicht vollen administrativen Zugriff auf Amazon Aurora DSQL über die AWS Management Console. Principals mit diesen Berechtigungen können Aurora DSQL-Cluster, einschließlich Clustern mit mehreren Regionen, mit der Konsole erstellen, löschen und aktualisieren. Sie können Cluster auflisten und Informationen zu einzelnen Clustern einsehen. Sie können Tags auf jeder Ressource in Ihrem Konto sehen. Sie können wie jeder andere Benutzer, auch der Administrator, eine Verbindung zur Datenbank herstellen. Sie können Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster ausführen, einschließlich des Startens, Stoppens und Überwachen von Sicherungs- und Wiederherstellungsaufträgen. Die Richtlinie umfasst AWS KMS Berechtigungen, die Operationen mit vom Kunden verwalteten Schlüsseln ermöglichen, die für die Cluster-Verschlüsselung verwendet werden. Sie können AWS CloudShell vom gestartet werden. AWS Management Console Sie können alle Metriken in CloudWatch Ihrem Konto sehen. Sie sind auch berechtigt, dienstbezogene Rollen für den `dsql.amazonaws.com` Service zu erstellen, was für die Erstellung von Clustern erforderlich ist.

Sie finden die `AmazonAuroraDSQLConsoleFullAccess` Richtlinie auf der IAM-Konsole und AmazonAuroraDSQLConsoleFullAccessim Referenzhandbuch für AWS verwaltete Richtlinien.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt volle Administratorrechte für alle Ressourcen in Aurora DSQL über die. AWS Management Console
- `cloudwatch`— erteilt die Berechtigung zum Abrufen von Batch-Mengen CloudWatch metrischer Daten und zum Durchführen metrischer Berechnungen mit abgerufenen Daten.
- `tag`— erteilt die Berechtigung, Tag-Schlüssel und -Werte zurückzugeben, die derzeit in dem AWS-Region für das aufrufende Konto angegebenen Tag verwendet werden.
- `backup and restore`— gewährt Berechtigungen zum Starten, Stoppen und Überwachen von Sicherungs- und Wiederherstellungsaufträgen für Aurora DSQL-Cluster.
- `kms`— gewährt die erforderlichen Berechtigungen, um den Zugriff auf vom Kunden verwaltete Schlüssel zu validieren, die für die Aurora DSQL-Clusterverschlüsselung beim Erstellen, Aktualisieren oder Herstellen einer Verbindung zu Clustern verwendet werden.
- `cloudshell`— gewährt Startberechtigungen für AWS CloudShell die Interaktion mit Aurora DSQL.
- `ec2`— erteilt die Erlaubnis, Amazon VPC-Endpunktinformationen einzusehen, die für Aurora DSQL-Verbindungen benötigt werden.

Sie finden die `AmazonAuroraDSQLReadOnlyAccess` Richtlinie auf der IAM-Konsole und AmazonAuroraDSQLReadOnlyAccessim AWS Managed Policy Reference Guide.

AWS verwaltete Richtlinie: Aurora DSQLService RolePolicy

Sie können Aurora nicht DSQLService RolePolicy an Ihre IAM-Entitäten anhängen. Diese Richtlinie ist mit einer serviceverknüpften Rolle verknüpft, die Aurora DSQL den Zugriff auf Kontoressourcen ermöglicht.

Sie finden die `AuroraDSQLServiceRolePolicy` Richtlinie auf der IAM-Konsole und DSQLService RolePolicy in Aurora im AWS Managed Policy Reference Guide.

Aurora DSQL-Updates für AWS verwaltete Richtlinien

Sehen Sie sich Details zu Aktualisierungen der AWS verwalteten Richtlinien für Aurora DSQL an, seit dieser Service begonnen hat, diese Änderungen zu verfolgen. Abonnieren Sie den RSS-Feed auf der Seite Aurora DSQL Document History, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderung	Beschreibung	Datum
AmazonAuroraDSQLFullAuf das Update zugreifen	Fügt die Möglichkeit hinzu, Sicherungs- und Wiederher stellungsvorgänge für Aurora DSQL-Cluster durchzufü hren, einschließlich Starten, Stoppen und Überwachen von Jobs. Es bietet auch die Möglichkeit, vom Kunden verwaltete KMS-Schlüssel für die Clusterverschlüsselung zu verwenden. Weitere Informationen finden Sie unter AmazonAur oraDSQLFullZugreifen und Verwenden von serviceve rknüpften Rollen in Aurora DSQL.	21. Mai 2025
AmazonAuroraDSQLCo nsoleFullAccess update	Fügt die Möglichkeit hinzu, Sicherungs- und Wiederher stellungsvorgänge für Aurora DSQL-Cluster über die AWS Console Home durchzuführen. Dies beinhaltet das Starten, Stoppen und Überwachen von Jobs. Es unterstützt auch die Verwendung von vom Kunden verwalteten KMS-Schlüsseln für die Clusterverschlüsselung und den Start AWS CloudShel l. Weitere Informationen finden Sie unter AmazonAur	21. Mai 2025

Änderung	Beschreibung	Datum
	oraDSQLConsoleFull Accessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	

Änderung	Beschreibung	Datum
AmazonAuroraDSQLFullZugriffsupdate	Die Richtlinie fügt vier neue Berechtigungen zum Erstellen und Verwalten von Datenbankclustern für mehrere AWS-Regionen:`PutMultiRegionProperties` , `PutWitnessRegion` `AddPeerCluster` , und hinzu`RemovePeerCluster` . Zu diesen Berechtigungen gehören Steuerungen auf Ressourcenebene und Bedingungsschlüssel, sodass Sie steuern können, welche Cluster-Benutzer Sie ändern können. Die Richtlinie fügt auch die `GetVpcEndpointServiceName` Erlaubnis hinzu, Ihnen dabei zu helfen, eine Verbindung zu Ihren Aurora DSQL-Clustern herzustellen. AWS PrivateLink Weitere Informationen finden Sie unter Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugreifen und Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025

Änderung	Beschreibung	Datum
AmazonAuroraDSQLReadOnlyAccess update	Beinhaltet die Möglichkeit, den richtigen VPC-Endpunkt-Servicenamen zu ermitteln, wenn über AWS PrivateLink Aurora DSQL eine Verbindung zu Ihren Aurora DSQL-Clustern hergestellt wird. Dadurch werden eindeutige Endpunkte pro Zelle erstellt, sodass diese API sicherstellt, dass Sie den richtigen Endpunkt für Ihren Cluster identifizieren und Verbindungsfehler vermeiden können. Weitere Informationen finden Sie unter AmazonAuroraDSQLReadOnlyAccessund Verwenden von servicevernüpften Rollen in Aurora DSQL.	13. Mai 2025

Änderung	Beschreibung	Datum
AmazonAuroraDSQLConsoleFullAccess update	Fügt Aurora DSQL neue Berechtigungen hinzu, um Clustermanagement in mehreren Regionen und VPC-Endpunktverbindungen zu unterstützen. Zu den neuen Berechtigungen gehören: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025

Änderung	Beschreibung	Datum
AuroraDsqlServiceLinkedRole Policy update	Fügt die Möglichkeit hinzu, Metriken in der Richtlinie `AWS/AuroraDSQL` und in `AWS/Usage` CloudWatch Namespaces zu veröffentlichen. Auf diese Weise kann der zugehörige Dienst oder die zugehörige Rolle umfassendere Nutzungs- und Leistungsdaten an Ihre Umgebung senden. CloudWatch Weitere Informationen finden Sie unter AuroraDsqlServiceLinkedRolePolicyund Verwenden von servicevorknüpften Rollen in Aurora DSQL.	8. Mai 2025
Seite wurde erstellt	Mit der Verfolgung AWS verwalteter Richtlinien im Zusammenhang mit Amazon Aurora DSQL wurde begonnen	3. Dezember 2024

Datenschutz in Amazon Aurora DSQL

Das Modell der gilt für den Datenschutz in. Wie in diesem Modell beschrieben, ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag -Modell der geteilten Verantwortung und in der DSGVO im -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder einrichten AWS Identity and Access Management. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von Pfaden zur Erfassung von Aktivitäten finden Sie unter Arbeiten mit Pfaden im Benutzerhandbuch.
- Verwenden Sie Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.

Wir empfehlen dringend, niemals vertrauliche oder sensible Informationen, wie z. B. die E-Mail-Adressen Ihrer Kunden, in Tags oder frei formatierte Textfelder wie ein Namensfeld einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der API oder oder auf andere Weise arbeiten oder AWS SDKs diese verwenden. AWS CLI Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Datenverschlüsselung

Amazon Aurora DSQL bietet eine äußerst robuste Speicherinfrastruktur, die für die Speicherung geschäftskritischer und primärer Daten konzipiert ist. Daten werden redundant auf mehreren Geräten in mehreren Einrichtungen in einer Aurora DSQL-Region gespeichert.

Verschlüsselung während der Übertragung

Standardmäßig ist die Verschlüsselung bei der Übertragung für Sie konfiguriert. Aurora DSQL verwendet TLS, um den gesamten Datenverkehr zwischen Ihrem SQL-Client und Aurora DSQL zu verschlüsseln.

Verschlüsselung und Signierung von Daten bei der Übertragung zwischen SDK AWS CLI- oder API-Clients und Aurora DSQL-Endpunkten:

- Aurora DSQL bietet HTTPS-Endpunkte für die Verschlüsselung von Daten während der Übertragung.
- Um die Integrität von API-Anfragen an Aurora DSQL zu schützen, müssen API-Aufrufe vom Aufrufer signiert werden. Anrufe werden mit einem X.509-Zertifikat oder dem AWS geheimen Zugriffsschlüssel des Kunden gemäß dem Signature Version 4-Signaturprozess (Sigv4) signiert. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 im Allgemeine AWS-Referenz.
- Verwenden Sie die AWS CLI oder eine der Optionen, um Anfragen AWS SDKs an zu stellen. AWS Diese Tools signieren automatisch die Anforderungen für Sie mit dem Zugriffsschlüssel, den Sie bei der Konfiguration der Tools angegeben haben.

Informationen zur Verschlüsselung im Ruhezustand finden Sie unterVerschlüsselung im Ruhezustand in Aurora DSQL.

Datenschutz für den Datenverkehr zwischen Netzwerken

Verbindungen sind sowohl zwischen Aurora DSQL und lokalen Anwendungen als auch zwischen Aurora DSQL und anderen AWS Ressourcen innerhalb derselben geschützt. AWS-Region

Sie haben zwei Verbindungsoptionen zwischen Ihrem privaten Netzwerk und: AWS

- Eine AWS Site-to-Site VPN-Verbindung. Weitere Informationen finden Sie unter Was ist AWS Site-to-Site VPN?
- Eine AWS Direct Connect Verbindung. Weitere Informationen finden Sie unter Was ist AWS Direct Connect?

Sie erhalten Zugriff auf Aurora DSQL über das Netzwerk, indem Sie AWS-published API-Operationen verwenden. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Konfiguration von SSL/TLS Zertifikaten für Aurora DSQL-Verbindungen

Aurora DSQL erfordert, dass alle Verbindungen die Transport Layer Security (TLS) -Verschlüsselung verwenden. Um sichere Verbindungen herzustellen, muss Ihr Client-System der Amazon Root Certificate Authority (Amazon Root CA 1) vertrauen. Dieses Zertifikat ist auf vielen Betriebssystemen vorinstalliert. Dieser Abschnitt enthält Anweisungen zur Überprüfung des vorinstallierten Amazon Root CA 1-Zertifikats auf verschiedenen Betriebssystemen und führt Sie durch den Prozess der manuellen Installation des Zertifikats, falls es noch nicht vorhanden ist.

Wir empfehlen die Verwendung von PostgreSQL Version 17.

> **Important**
>
> Für Produktionsumgebungen empfehlen wir die Verwendung des `verify-full` SSL-Modus, um ein Höchstmaß an Verbindungssicherheit zu gewährleisten. In diesem Modus wird überprüft, ob das Serverzertifikat von einer vertrauenswürdigen Zertifizierungsstelle signiert ist und ob der Server-Hostname mit dem Zertifikat übereinstimmt.

Überprüfung der vorinstallierten Zertifikate

In den meisten Betriebssystemen ist Amazon Root CA 1 bereits vorinstalliert. Um dies zu überprüfen, können Sie die folgenden Schritte ausführen.

Linux (RedHat/CentOS/Fedora)

Führen Sie den folgenden Befehl in Ihrem Terminal aus:

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

Wenn das Zertifikat installiert ist, sehen Sie die folgende Ausgabe:

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

macOS

1. Öffnen Sie die Spotlight-Suche (Befehlstaste+Leertaste)
2. Suchen Sie nach Keychain Access
3. Wählen Sie unter Systemschlüsselanhänger die Option System Roots aus

4. Suchen Sie in der Zertifikatsliste nach Amazon Root CA 1

Windows

> **Note**
>
> Aufgrund eines bekannten Problems mit dem PSQL-Windows-Client kann bei der Verwendung von Systemstammzertifikaten (`sslrootcert=system`) der folgende Fehler zurückgegeben werden:`SSL error: unregistered scheme`. Sie können das Verbindung von Windows aus herstellen als alternative Methode verwenden, um mithilfe von SSL eine Verbindung zu Ihrem Cluster herzustellen.

Wenn Amazon Root CA 1 nicht in Ihrem Betriebssystem installiert ist, gehen Sie wie folgt vor.

Zertifikate installieren

Wenn das `Amazon Root CA 1` Zertifikat nicht auf Ihrem Betriebssystem vorinstalliert ist, müssen Sie es manuell installieren, um sichere Verbindungen zu Ihrem Aurora DSQL-Cluster herzustellen.

Installation des Linux-Zertifikats

Gehen Sie wie folgt vor, um das Amazon Root CA-Zertifikat auf Linux-Systemen zu installieren.

1. Laden Sie das Root-Zertifikat herunter:

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

2. Kopieren Sie das Zertifikat in den Trust Store:

```
sudo cp ./AmazonRootCA1.pem /etc/pki/ca-trust/source/anchors/
```

3. Aktualisieren Sie den CA Trust Store:

```
sudo update-ca-trust
```

4. Überprüfen Sie die Installation:

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

Installation des macOS-Zertifikats

Diese Schritte zur Installation des Zertifikats sind optional. Sie funktionieren Installation des Linux-Zertifikats auch für ein MacOS.

1. Laden Sie das Root-Zertifikat herunter:

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

2. Fügen Sie das Zertifikat dem Systemschlüsselbund hinzu:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain
 AmazonRootCA1.pem
```

3. Überprüfen Sie die Installation:

```
security find-certificate -a -c "Amazon Root CA 1" -p /Library/Keychains/
System.keychain
```

Verbindung wird mit SSL/TLS Bestätigung hergestellt

Bevor Sie SSL/TLS Zertifikate für sichere Verbindungen zu Ihrem Aurora DSQL-Cluster konfigurieren, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen.

- PostgreSQL Version 17 installiert
- AWS CLI mit den entsprechenden Anmeldeinformationen konfiguriert
- Informationen zum Aurora DSQL-Cluster-Endpunkt

Von Linux aus wird eine Verbindung hergestellt

1. Generieren und legen Sie das Authentifizierungstoken fest:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-
cluster-region --hostname your-cluster-endpoint)
```

2. Stellen Sie mithilfe von Systemzertifikaten eine Verbindung her (falls vorinstalliert):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
```

```
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Oder stellen Sie mithilfe eines heruntergeladenen Zertifikats eine Verbindung her:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

> **Note**
>
> Weitere Informationen zu den PGSSLMODE-Einstellungen finden Sie unter sslmode in der Dokumentation zu den Funktionen zur Datenbankverbindungssteuerung von PostgreSQL 17.

Von macOS aus verbinden

1. Generieren und legen Sie das Authentifizierungstoken fest:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Stellen Sie mithilfe von Systemzertifikaten eine Verbindung her (falls vorinstalliert):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Oder laden Sie das Stammzertifikat herunter und speichern Sie es unter `root.pem` (falls das Zertifikat nicht vorinstalliert ist)

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
```

```
--host your_cluster_endpoint
```

4. Connect mit psql her:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

Verbindung von Windows aus herstellen

Verwenden der Befehlszeile

1. Generieren Sie das Authentifizierungstoken:

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

2. Legen Sie die Umgebungsvariable für das Passwort fest:

```
set "PGPASSWORD=token-from-above"
```

3. Stellen Sie die SSL-Konfiguration ein:

```
set PGSSLROOTCERT=C:\full\path\to\root.pem
set PGSSLMODE=verify-full
```

4. Connect zur Datenbank her:

```
"C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres ^
--username admin ^
--host your-cluster-endpoint
```

Benutzen PowerShell

1. Generieren und legen Sie das Authentifizierungstoken fest:

```
$env:PGPASSWORD = (aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --expires-in=3600 --hostname=your-cluster-endpoint)
```

2. Stellen Sie die SSL-Konfiguration ein:

```
$env:PGSSLROOTCERT='C:\full\path\to\root.pem'
$env:PGSSLMODE='verify-full'
```

3. Connect zur Datenbank her:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
--username admin `
--host your-cluster-endpoint
```

Weitere Ressourcen

- PostgreSQL SSL-Dokumentation
- Amazon Trust Services

Datenverschlüsselung für Amazon Aurora DSQL

Amazon Aurora DSQL verschlüsselt alle Benutzerdaten im Ruhezustand. Um die Sicherheit zu erhöhen, verwendet diese Verschlüsselung AWS Key Management Service ()AWS KMS. Diese Funktionalität trägt zur Verringerung des Betriebsaufwands und der Komplexität bei, die mit dem Schutz sensibler Daten einhergeht. Verschlüsselung im Ruhezustand hilft Ihnen:

- Reduzieren Sie den betrieblichen Aufwand, der durch den Schutz sensibler Daten entsteht
- Entwickeln Sie sicherheitsrelevante Anwendungen, die strenge Verschlüsselungsvorschriften und regulatorische Anforderungen erfüllen
- Fügen Sie eine zusätzliche Datenschutzebene hinzu, indem Sie Ihre Daten stets in einem verschlüsselten Cluster sichern
- Halten Sie Unternehmensrichtlinien, Branchen- oder behördliche Vorschriften und Compliance-Anforderungen ein

Mit Aurora DSQL können Sie sicherheitsrelevante Anwendungen entwickeln, die strenge Verschlüsselungsvorschriften und regulatorische Anforderungen erfüllen. In den folgenden Abschnitten wird erklärt, wie Sie die Verschlüsselung für neue und bestehende Aurora DSQL-Datenbanken konfigurieren und Ihre Verschlüsselungsschlüssel verwalten.

Themen

KMS-Schlüsseltypen für Aurora DSQL

Aurora DSQL lässt sich integrieren AWS KMS , um die Verschlüsselungsschlüssel für Ihre Cluster zu verwalten. Weitere Informationen zu Schlüsseltypen und Status finden Sie unter AWS Key Management Service Konzepte im AWS Key Management Service Entwicklerhandbuch. Wenn Sie einen neuen Cluster erstellen, können Sie aus den folgenden KMS-Schlüsseltypen wählen, um Ihren Cluster zu verschlüsseln:

AWS-eigener Schlüssel

Standardverschlüsselungstyp. Aurora DSQL besitzt den Schlüssel ohne zusätzliche Kosten für Sie. Amazon Aurora DSQL entschlüsselt Cluster-Daten transparent, wenn Sie auf einen verschlüsselten Cluster zugreifen. Sie müssen Ihren Code oder Ihre Anwendungen nicht ändern, um verschlüsselte Cluster zu verwenden oder zu verwalten, und alle Aurora DSQL-Abfragen funktionieren mit Ihren verschlüsselten Daten.

Kundenverwalteter Schlüssel

Sie erstellen, besitzen und verwalten den Schlüssel in Ihrem AWS-Konto. Sie haben die volle Kontrolle über den KMS-Schlüssel. AWS KMS Es fallen Gebühren an.

Die Verschlüsselung im Ruhezustand mit der AWS-eigener Schlüssel ist ohne zusätzliche Kosten verfügbar. Für vom Kunden verwaltete Schlüssel fallen jedoch AWS KMS Gebühren an. Weitere Informationen finden Sie in der AWS KMS-Preisliste.

Sie können jederzeit zwischen diesen Schlüsseltypen wechseln. Weitere Informationen zu Schlüsseltypen finden Sie unter Vom Kunden verwaltete Schlüssel und AWS-eigene Schlüsselim AWS Key Management Service Entwicklerhandbuch.

> **Note**
>
> Aurora DSQL Encryption at Rest ist in allen AWS Regionen verfügbar, in denen Aurora DSQL verfügbar ist.

Verschlüsselung im Ruhezustand in Aurora DSQL

Amazon Aurora DSQL verwendet den 256-Bit-Advanced Encryption Standard (AES-256), um Ihre Daten im Ruhezustand zu verschlüsseln. Diese Verschlüsselung trägt dazu bei, Ihre Daten vor unbefugtem Zugriff auf den zugrunde liegenden Speicher zu schützen. AWS KMS verwaltet die Verschlüsselungsschlüssel für Ihre Cluster. Sie können den Standard AWS-eigene Schlüssel verwenden oder Ihren eigenen wählen AWS KMS Kundenverwaltete Schlüssel. Weitere Informationen zur Angabe und Verwaltung von Schlüsseln für Ihre Aurora DSQL-Cluster finden Sie unter Einen verschlüsselten Aurora DSQL-Cluster erstellen undEinen Schlüssel für Ihren Aurora DSQL-Cluster entfernen oder aktualisieren.

Themen

- AWS-eigene Schlüssel
- Kundenverwaltete Schlüssel

AWS-eigene Schlüssel

Aurora DSQL verschlüsselt standardmäßig alle Cluster mit. AWS-eigene Schlüssel Diese Schlüssel können kostenlos verwendet werden und werden jährlich gewechselt, um Ihre Kontoressourcen zu schützen. Sie müssen diese Schlüssel nicht einsehen, verwalten, verwenden oder prüfen, sodass keine Maßnahmen zum Datenschutz erforderlich sind. Weitere Informationen zu AWS-eigene Schlüssel finden Sie AWS-eigene Schlüsselim AWS Key Management Service Entwicklerhandbuch.

Kundenverwaltete Schlüssel

Sie erstellen, besitzen und verwalten von Kunden verwaltete Schlüssel in Ihrem AWS-Konto. Sie haben die volle Kontrolle über diese KMS-Schlüssel, einschließlich ihrer Richtlinien, Verschlüsselungsmaterialien, Tags und Aliase. Weitere Informationen zur Verwaltung von Berechtigungen finden Sie unter Vom Kunden verwaltete Schlüssel im AWS Key Management Service Entwicklerhandbuch.

Wenn Sie einen vom Kunden verwalteten Schlüssel für die Verschlüsselung auf Clusterebene angeben, verschlüsselt Aurora DSQL den Cluster und alle seine regionalen Daten mit diesem Schlüssel. Um Datenverlust zu verhindern und den Cluster-Zugriff aufrechtzuerhalten, benötigt Aurora DSQL Zugriff auf Ihren Verschlüsselungsschlüssel. Wenn Sie Ihren vom Kunden verwalteten Schlüssel deaktivieren, Ihren Schlüssel für die Löschung planen oder eine Richtlinie haben, die Ihren Servicezugriff einschränkt, ändert sich der Verschlüsselungsstatus für Ihren Cluster auf. `KMS_KEY_INACCESSIBLE` Wenn Aurora DSQL nicht auf den Schlüssel zugreifen kann, können Benutzer keine Verbindung zum Cluster herstellen, der Verschlüsselungsstatus für den Cluster ändert sich `KMS_KEY_INACCESSIBLE` und der Service verliert den Zugriff auf die Clusterdaten.

Bei Clustern mit mehreren Regionen können Kunden den AWS KMS Verschlüsselungsschlüssel jeder Region separat konfigurieren, und jeder regionale Cluster verwendet seinen eigenen Verschlüsselungsschlüssel auf Clusterebene. Wenn Aurora DSQL nicht auf den Verschlüsselungsschlüssel für einen Peer in einem Cluster mit mehreren Regionen zugreifen kann, wird der Status für diesen Peer geändert `KMS_KEY_INACCESSIBLE` und er ist für Lese- und Schreibvorgänge nicht mehr verfügbar. Andere Peers setzen den normalen Betrieb fort.

> **Note**
>
> Wenn Aurora DSQL nicht auf Ihren vom Kunden verwalteten Schlüssel zugreifen kann, ändert sich Ihr Cluster-Verschlüsselungsstatus auf`KMS_KEY_INACCESSIBLE`. Nachdem Sie den Schlüsselzugriff wiederhergestellt haben, erkennt der Service die Wiederherstellung automatisch innerhalb von 15 Minuten. Weitere Informationen finden Sie unter Cluster-Leerlauf.
>
> Wenn bei Clustern mit mehreren Regionen der Schlüsselzugriff über einen längeren Zeitraum verloren geht, hängt die Clusterwiederherstellungszeit davon ab, wie viele Daten geschrieben wurden, während auf den Schlüssel nicht zugegriffen werden konnte.

Verwendung AWS KMS und Datenschlüssel mit Aurora DSQL

Die Aurora-DSQL-Verschlüsselung im Ruhezustand verwendet eine AWS KMS key und eine Hierarchie von Datenschlüsseln, um Ihre Clusterdaten zu schützen.

Wir empfehlen Ihnen, Ihre Verschlüsselungsstrategie zu planen, bevor Sie Ihren Cluster in Aurora DSQL implementieren. Wenn Sie sensible oder vertrauliche Daten in Aurora DSQL speichern, sollten Sie erwägen, die clientseitige Verschlüsselung in Ihren Plan aufzunehmen. Auf diese Weise können Sie Daten so nah wie möglich an ihrem Ursprung verschlüsseln und den Schutz der Daten während ihres gesamten Lebenszyklus gewährleisten.

Themen

Verwenden von AWS KMS key s mit Aurora DSQL

Verschlüsselung im Ruhezustand schützt Ihren Aurora DSQL-Cluster unter einem AWS KMS key. Standardmäßig verwendet Aurora DSQL einen Mehrmandanten-Verschlüsselungsschlüssel AWS-eigener Schlüssel, der in einem Aurora DSQL-Dienstkonto erstellt und verwaltet wird. Sie können Ihre Aurora DSQL-Cluster jedoch unter einem vom Kunden verwalteten Schlüssel in Ihrem verschlüsseln. AWS-Konto Sie können für jeden Cluster einen anderen KMS-Schlüssel auswählen, auch wenn dieser an einer Einrichtung mit mehreren Regionen teilnimmt.

Sie wählen den KMS-Schlüssel für einen Cluster aus, wenn Sie den Cluster erstellen oder aktualisieren. Sie können den KMS-Schlüssel für einen Cluster jederzeit ändern, entweder in der Aurora DSQL-Konsole oder mithilfe des `UpdateCluster` Vorgangs. Der Vorgang des Schlüsselwechsels erfordert keine Ausfallzeiten oder Leistungseinbußen.

> **Important**
>
> Aurora DSQL unterstützt nur symmetrische KMS-Schlüssel. Sie können keinen asymmetrischen KMS-Schlüssel verwenden, um Ihre Aurora DSQL-Cluster zu verschlüsseln.

Ein vom Kunden verwalteter Schlüssel bietet die folgenden Vorteile.

- Sie erstellen und verwalten den KMS-Schlüssel, einschließlich der Festlegung der Schlüsselrichtlinien und IAM-Richtlinien zur Steuerung des Zugriffs auf den KMS-Schlüssel. Sie können den KMS-Schlüssel aktivieren und deaktivieren, die automatische Schlüsseldrehung aktivieren und deaktivieren und den KMS-Schlüssel löschen, wenn er nicht mehr verwendet wird.
- Sie können einen kundenverwalteten Schlüssel mit importiertem Schlüsselmaterial oder einen kundenverwalteten Schlüssel in einem benutzerdefinierten Schlüsselspeicher verwenden, den Sie besitzen und verwalten.
- Sie können die Verschlüsselung und Entschlüsselung Ihres Aurora DSQL-Clusters überprüfen, indem Sie die Aurora-DSQL-API-Aufrufe in den Protokollen untersuchen. AWS KMS AWS CloudTrail

Sie AWS-eigener Schlüssel ist jedoch kostenlos und ihre Nutzung wird nicht auf AWS KMS Ressourcen- oder Anforderungskontingente angerechnet. Für vom Kunden verwaltete Schlüssel fällt für jeden API-Aufruf eine Gebühr an, und für diese Schlüssel gelten AWS KMS Kontingente.

Clusterschlüssel mit Aurora DSQL verwenden

Aurora DSQL verwendet den AWS KMS key für den Cluster, um einen eindeutigen Datenschlüssel für den Cluster, den so genannten Clusterschlüssel, zu generieren und zu verschlüsseln.

Der Clusterschlüssel wird als Schlüssel zur Verschlüsselung verwendet. Aurora DSQL verwendet diesen Clusterschlüssel, um Datenverschlüsselungsschlüssel zu schützen, die zur Verschlüsselung der Clusterdaten verwendet werden. Aurora DSQL generiert einen eindeutigen Datenverschlüsselungsschlüssel für jede zugrunde liegende Struktur in einem Cluster, aber mehrere Clusterelemente können durch denselben Datenverschlüsselungsschlüssel geschützt werden.

Um den Clusterschlüssel zu entschlüsseln, sendet Aurora DSQL eine Anfrage an, AWS KMS wenn Sie zum ersten Mal auf einen verschlüsselten Cluster zugreifen. Um den Cluster verfügbar zu halten, überprüft Aurora DSQL regelmäßig den Entschlüsselungszugriff auf den KMS-Schlüssel, auch wenn Sie nicht aktiv auf den Cluster zugreifen.

Aurora DSQL speichert und verwendet den Clusterschlüssel und die Datenverschlüsselungsschlüssel außerhalb von AWS KMS. Alle Schlüssel werden mit Advanced Encryption Standard (AES)-Verschlüsselung und 256-Bit-Verschlüsselungsschlüsseln geschützt. Anschließend werden die verschlüsselten Schlüssel zusammen mit den verschlüsselten Daten gespeichert, sodass sie bei Bedarf für die Entschlüsselung der Clusterdaten zur Verfügung stehen.

Wenn Sie den KMS-Schlüssel für Ihren Cluster ändern, verschlüsselt Aurora DSQL den vorhandenen Clusterschlüssel erneut mit dem neuen KMS-Schlüssel.

Zwischenspeichern von Cluster-Schlüsseln

Um zu vermeiden, dass jede Aurora-DSQL-Operation aufgerufen AWS KMS wird, speichert Aurora DSQL die Klartext-Clusterschlüssel für jeden Aufrufer im Speicher zwischen. Wenn Aurora DSQL nach 15 Minuten Inaktivität eine Anfrage für den zwischengespeicherten Clusterschlüssel erhält, sendet es eine neue Anfrage an, um den Clusterschlüssel AWS KMS zu entschlüsseln. Dieser Aufruf erfasst alle Änderungen, die nach der letzten Anfrage zur Entschlüsselung des AWS KMS key Clusterschlüssels an den Zugriffsrichtlinien von In AWS KMS oder AWS Identity and Access Management (IAM) vorgenommen wurden.

Autorisieren der Verwendung Ihres AWS KMS key für Aurora DSQL

Wenn Sie in Ihrem Konto einen vom Kunden verwalteten Schlüssel verwenden, um Ihren Aurora DSQL-Cluster zu schützen, müssen die Richtlinien für diesen Schlüssel Aurora DSQL die Erlaubnis geben, ihn in Ihrem Namen zu verwenden.

Sie haben die volle Kontrolle über die Richtlinien für einen vom Kunden verwalteten Schlüssel. Aurora DSQL benötigt keine zusätzliche Autorisierung, um die Standardeinstellung AWS-eigener Schlüssel zum Schutz der Aurora DSQL-Cluster in Ihrem zu verwenden. AWS-Konto

Schlüsselrichtlinie für einen kundenverwalteten Schlüssel

Wenn Sie einen vom AWS KMS key Kunden verwalteten Schlüssel zum Schutz eines Aurora DSQL-Clusters auswählen, benötigt Aurora DSQL die Erlaubnis, den im Namen des Prinzipals zu verwenden, der die Auswahl trifft. Dieser Prinzipal, ein Benutzer oder eine Rolle, muss über die Berechtigungen verfügen AWS KMS key , die Aurora DSQL benötigt. Sie können diese Berechtigungen in einer Schlüsselrichtlinie oder einer IAM-Richtlinie bereitstellen.

Aurora DSQL benötigt mindestens die folgenden Berechtigungen für einen vom Kunden verwalteten Schlüssel:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(für kms: ReEncryptFrom und kms:ReEncryptTo)
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Beispielsweise bietet die folgende Beispiel-Schlüsselrichtlinie nur die erforderlichen Berechtigungen. Die Richtlinie hat folgende Auswirkungen:

- Ermöglicht Aurora DSQL die Verwendung von AWS KMS key in kryptografischen Vorgängen, jedoch nur, wenn es im Namen von Prinzipalen im Konto handelt, die die Erlaubnis haben, Aurora DSQL zu verwenden. Wenn die in der Richtlinienerklärung angegebenen Principals nicht berechtigt sind, Aurora DSQL zu verwenden, schlägt der Anruf fehl, auch wenn er vom Aurora DSQL-Dienst stammt.
- Der `kms:ViaService` Bedingungsschlüssel erlaubt die Berechtigungen nur, wenn die Anfrage von Aurora DSQL im Namen der in der Grundsatzerklärung aufgeführten Prinzipale stammt. Diese Prinzipale können diese Operationen nicht direkt aufrufen.
- Gewährt den AWS KMS key Administratoren (Benutzern, die die `db-team` Rolle übernehmen können) schreibgeschützten Zugriff auf AWS KMS key

Bevor Sie ein Beispiel für eine Schlüsselrichtlinie verwenden, ersetzen Sie die Beispielprinzipale durch tatsächliche Prinzipale aus Ihrem. AWS-Konto

```
{
  "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"
    }
  }
}
```

Aurora DSQL-Verschlüsselungskontext

Ein Verschlüsselungskontext ist eine Gruppe von Schlüssel/Wert-Paaren mit willkürlichen, nicht geheimen Daten. Wenn Sie einen Verschlüsselungskontext in eine Anforderung zur Verschlüsselung von Daten aufnehmen, wird der Verschlüsselungskontext AWS KMS kryptografisch an die verschlüsselten Daten gebunden. Zur Entschlüsselung der Daten müssen Sie denselben Verschlüsselungskontext übergeben.

Aurora DSQL verwendet bei allen AWS KMS kryptografischen Vorgängen denselben Verschlüsselungskontext. Wenn Sie einen vom Kunden verwalteten Schlüssel zum Schutz Ihres Aurora DSQL-Clusters verwenden, können Sie den Verschlüsselungskontext verwenden, um die Verwendung des AWS KMS key in Auditaufzeichnungen und Protokollen zu identifizieren. Er erscheint auch im Klartext in Protokollen wie denen in. AWS CloudTrail

Der Verschlüsselungskontext kann auch als Bedingung für die Autorisierung in Richtlinien verwendet werden.

In seinen Anfragen an AWS KMS verwendet Aurora DSQL einen Verschlüsselungskontext mit einem Schlüssel-Wert-Paar:

```
"encryptionContext": {
  "aws:dsql:ClusterId": "w4abucpbwuxx"
},
```

Das Schlüssel-Wert-Paar identifiziert den Cluster, den Aurora DSQL verschlüsselt. Der Schlüssel lautet `aws:dsql:ClusterId`. Der Wert ist der Identifier des Clusters.

Überwachung der Aurora DSQL-Interaktion mit AWS KMS

Wenn Sie einen vom Kunden verwalteten Schlüssel zum Schutz Ihrer Aurora DSQL-Cluster verwenden, können Sie AWS CloudTrail Protokolle verwenden, um die Anfragen zu verfolgen, an die Aurora DSQL in AWS KMS Ihrem Namen sendet.

Erweitern Sie die folgenden Abschnitte, um zu erfahren, wie Aurora DSQL die AWS KMS Operationen `GenerateDataKey` und `Decrypt` verwendet.

`GenerateDataKey`

Wenn Sie die Verschlüsselung im Ruhezustand auf einem Cluster aktivieren, erstellt Aurora DSQL einen eindeutigen Clusterschlüssel. Es sendet eine `GenerateDataKey` Anfrage an AWS KMS , die den AWS KMS key für den Cluster spezifiziert.

Das Ereignis, das die `GenerateDataKey`-Operation aufzeichnet, ähnelt dem folgenden Beispielereignis. Der Benutzer ist das Aurora DSQL-Dienstkonto. Zu den Parametern gehören der Amazon-Ressourcenname (ARN) von AWS KMS key, ein Schlüsselspezifizierer, für den ein 256-Bit-Schlüssel erforderlich ist, und der Verschlüsselungskontext, der den Cluster identifiziert.

```
{
    "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

Wenn Sie auf einen verschlüsselten Aurora DSQL-Cluster zugreifen, muss Aurora DSQL den Clusterschlüssel entschlüsseln, damit es die Schlüssel entschlüsseln kann, die sich in der Hierarchie darunter befinden. Anschließend werden die Daten im Cluster entschlüsselt. Um den Clusterschlüssel zu entschlüsseln, sendet Aurora DSQL eine `Decrypt` Anfrage an AWS KMS , die den AWS KMS key für den Cluster spezifiziert.

Das Ereignis, das die `Decrypt`-Operation aufzeichnet, ähnelt dem folgenden Beispielereignis. Der Benutzer ist Ihr Hauptbenutzer AWS-Konto , der auf den Cluster zugreift. Zu den Parametern gehören der verschlüsselte Clusterschlüssel (als Chiffretext-Blob) und der Verschlüsselungskontext, der den Cluster identifiziert. AWS KMS leitet die ID von aus dem Chiffretext ab. 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"
}
```


Einen verschlüsselten Aurora DSQL-Cluster erstellen

Alle Aurora DSQL-Cluster sind im Ruhezustand verschlüsselt. Standardmäßig verwenden Cluster einen kostenlosen AWS-eigener Schlüssel Schlüssel, oder Sie können einen benutzerdefinierten AWS KMS Schlüssel angeben. Gehen Sie wie folgt vor, um Ihren verschlüsselten Cluster entweder aus dem AWS Management Console oder dem zu erstellen AWS CLI.

Console

Um einen verschlüsselten Cluster im zu erstellen AWS Management Console

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Aurora DSQL-Konsole unter https://console.aws.amazon.com/dsql/.
2. Wählen Sie im Navigationsbereich auf der linken Seite der Konsole Clusters aus.
3. Wählen Sie oben rechts „Cluster erstellen“ und wählen Sie „Einzelne Region“ aus.
4. Wählen Sie in den Einstellungen für die Cluster-Verschlüsselung eine der folgenden Optionen aus.
   - Akzeptieren Sie die Standardeinstellungen für die Verschlüsselung ohne zusätzliche AWS-eigener Schlüssel Kosten.
   - Wählen Sie Verschlüsselungseinstellungen anpassen (erweitert), um einen benutzerdefinierten KMS-Schlüssel anzugeben. Suchen Sie dann nach der ID oder dem Alias Ihres KMS-Schlüssels oder geben Sie ihn ein. Wählen Sie alternativ Create an AWS KMS Key aus, um einen neuen Schlüssel in der AWS KMS Konsole zu erstellen.
5. Wählen Sie Cluster erstellen.

Um den Verschlüsselungstyp für Ihren Cluster zu bestätigen, navigieren Sie zur Seite Cluster und wählen Sie die ID des Clusters aus, um die Cluster-Details anzuzeigen. Überprüfen Sie die Registerkarte Cluster-Einstellungen. Die Cluster-KMS-Schlüsseleinstellung zeigt den Aurora DSQL-Standardschlüssel für Cluster, die AWS eigene Schlüssel verwenden, oder die Schlüssel-ID für andere Verschlüsselungstypen.

> **Note**
>
> Wenn Sie Ihren eigenen Schlüssel besitzen und verwalten möchten, stellen Sie sicher, dass Sie die KMS-Schlüsselrichtlinie entsprechend festlegen. Beispiele und weitere Informationen finden Sie unterthe section called “Schlüsselrichtlinie für einen kundenverwalteten Schlüssel”.

CLI

So erstellen Sie einen Cluster, der mit dem Standard verschlüsselt ist AWS-eigener Schlüssel

- Verwenden Sie den folgenden Befehl, um einen Aurora DSQL-Cluster zu erstellen.

```
aws dsql create-cluster
```

Wie in den folgenden Verschlüsselungsdetails gezeigt, ist der Verschlüsselungsstatus für den Cluster standardmäßig aktiviert, und der Standardverschlüsselungstyp ist AWS-eigener Schlüssel. Der Cluster ist jetzt mit dem standardmäßigen AWS-eigenen Schlüssel im Aurora DSQL-Servicekonto verschlüsselt.

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

Um einen Cluster zu erstellen, der mit Ihrem vom Kunden verwalteten Schlüssel verschlüsselt ist

- Verwenden Sie den folgenden Befehl, um einen Aurora DSQL-Cluster zu erstellen. Ersetzen Sie dabei die Schlüssel-ID in rotem Text durch die ID Ihres vom Kunden verwalteten Schlüssels.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Wie in den folgenden Verschlüsselungsdetails gezeigt, ist der Verschlüsselungsstatus für den Cluster standardmäßig aktiviert, und der Verschlüsselungstyp ist vom Kunden verwalteter KMS-Schlüssel. Der Cluster ist jetzt mit Ihrem Schlüssel verschlüsselt.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Einen Schlüssel für Ihren Aurora DSQL-Cluster entfernen oder aktualisieren

Sie können das AWS Management Console oder das verwenden AWS CLI , um die Verschlüsselungsschlüssel auf vorhandenen Clustern in Amazon Aurora DSQL zu aktualisieren oder zu entfernen. Wenn Sie einen Schlüssel entfernen, ohne ihn zu ersetzen, verwendet Aurora DSQL den Standard AWS-eigener Schlüssel. Gehen Sie wie folgt vor, um die Verschlüsselungsschlüssel eines vorhandenen Clusters über die Aurora DSQL-Konsole oder die AWS CLI zu aktualisieren.

Console

Um einen Verschlüsselungsschlüssel zu aktualisieren oder zu entfernen AWS Management Console

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Aurora DSQL-Konsole unter https://console.aws.amazon.com/dsql/.
2. Wählen Sie im Navigationsbereich auf der linken Seite der Konsole Clusters aus.
3. Suchen Sie in der Listenansicht die Zeile des Clusters, den Sie aktualisieren möchten, und wählen Sie sie aus.
4. Wählen Sie das Menü Aktionen und dann Ändern aus.
5. Wählen Sie in den Cluster-Verschlüsselungseinstellungen eine der folgenden Optionen, um Ihre Verschlüsselungseinstellungen zu ändern.
   - Wenn Sie von einem benutzerdefinierten Schlüssel zu einem wechseln möchten AWS-eigener Schlüssel, deaktivieren Sie die Option Verschlüsselungseinstellungen anpassen (erweitert). Die Standardeinstellungen werden angewendet und Ihr Cluster wird kostenlos verschlüsselt. AWS-eigener Schlüssel
   - Wenn Sie von einem benutzerdefinierten KMS-Schlüssel zu einem anderen oder von einem AWS-eigener Schlüssel zu einem KMS-Schlüssel wechseln möchten, wählen Sie die Option Verschlüsselungseinstellungen anpassen (erweitert), sofern sie nicht bereits ausgewählt ist. Suchen Sie dann nach der ID oder dem Alias des Schlüssels, den Sie verwenden möchten, und wählen Sie sie aus. Wählen Sie alternativ Create an AWS KMS Key, um einen neuen Schlüssel in der AWS KMS Konsole zu erstellen.
6. Wählen Sie Speichern.

CLI

Die folgenden Beispiele zeigen, wie Sie den verwenden AWS CLI , um einen verschlüsselten Cluster zu aktualisieren.

Um einen verschlüsselten Cluster mit der Standardeinstellung zu aktualisieren AWS-eigener Schlüssel

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

Der `EncryptionStatus` Wert der Clusterbeschreibung ist auf gesetzt `ENABLED` und der `EncryptionType` ist`AWS_OWNED_KMS_KEY`.

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

Dieser Cluster ist jetzt mit der Standardeinstellung AWS-eigener Schlüssel im Aurora DSQL-Dienstkonto verschlüsselt.

Um einen verschlüsselten Cluster mit einem vom Kunden verwalteten Schlüssel für Aurora DSQL zu aktualisieren

Aktualisieren Sie den verschlüsselten Cluster wie im folgenden Beispiel:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

Die `EncryptionStatus` Clusterbeschreibung geht über zu `UPDATING` und die `EncryptionType` ist`CUSTOMER_MANAGED_KMS_KEY`. Nachdem Aurora DSQL die Verbreitung

des neuen Schlüssels über die Plattform abgeschlossen hat, wird der Verschlüsselungsstatus auf geändert `ENABLED`

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Wenn Sie sich dafür entscheiden, Ihren eigenen Schlüssel zu besitzen und zu verwalten, stellen Sie sicher, dass Sie die KMS-Schlüsselrichtlinie entsprechend festlegen. Beispiele und weitere Informationen finden Sie unterthe section called “Schlüsselrichtlinie für einen kundenverwalteten Schlüssel”.

Überlegungen zur Verschlüsselung mit Aurora DSQL

- Aurora DSQL verschlüsselt alle Clusterdaten im Ruhezustand. Sie können diese Verschlüsselung nicht deaktivieren oder nur einige Elemente in einem Cluster verschlüsseln.
- AWS Backup verschlüsselt Ihre Backups und alle Cluster, die aus diesen Backups wiederhergestellt wurden. Sie können Ihre Backup-Daten entweder AWS Backup mit dem AWS eigenen Schlüssel oder mit einem vom Kunden verwalteten Schlüssel verschlüsseln.
- Die folgenden Datenschutzstatus sind für Aurora DSQL aktiviert:
  - Daten im Ruhezustand — Aurora DSQL verschlüsselt alle statischen Daten auf persistenten Speichermedien
  - Daten während der Übertragung — Aurora DSQL verschlüsselt die gesamte Kommunikation standardmäßig mit Transport Layer Security (TLS)
- Wenn Sie zu einem anderen Schlüssel wechseln, empfehlen wir, den ursprünglichen Schlüssel aktiviert zu lassen, bis der Übergang abgeschlossen ist. AWS benötigt den Originalschlüssel zum Entschlüsseln von Daten, bevor Ihre Daten mit dem neuen Schlüssel verschlüsselt werden. Der Vorgang ist abgeschlossen, wenn der Cluster aktiviert `encryptionStatus` ist `ENABLED` und Sie den Schlüssel `kmsKeyArn` des neuen, vom Kunden verwalteten Schlüssel sehen.

- Wenn Sie Ihren vom Kunden verwalteten Schlüssel deaktivieren oder Aurora DSQL den Zugriff auf die Verwendung Ihres Schlüssels entziehen, wechselt Ihr Cluster in den `IDLE` Status.
- Die DSQL-API AWS Management Console und die Amazon Aurora DSQL API verwenden unterschiedliche Begriffe für Verschlüsselungstypen:
  - AWS Konsole — In der Konsole sehen Sie, `KMS` wann Sie einen vom Kunden verwalteten Schlüssel verwenden und `DEFAULT` wann Sie einen AWS-eigener Schlüssel verwenden.
  - API — Die Amazon Aurora DSQL API verwendet `CUSTOMER_MANAGED_KMS_KEY` für vom Kunden verwaltete Schlüssel und `AWS_OWNED_KMS_KEY` für AWS-eigene Schlüssel.
- Wenn Sie bei der Clustererstellung keinen Verschlüsselungsschlüssel angeben, verschlüsselt Aurora DSQL Ihre Daten automatisch mit dem. AWS-eigener Schlüssel
- Sie können jederzeit zwischen einem AWS-eigener Schlüssel und einem vom Kunden verwalteten Schlüssel wechseln. Nehmen Sie diese Änderung mithilfe der AWS Management Console AWS CLI, oder der Amazon Aurora SQL API vor.

Identitäts- und Zugriffsmanagement für Aurora DSQL

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu kontrollieren. AWS IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Aurora DSQL-Ressourcen zu verwenden. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, die Sie in Aurora DSQL ausführen.

Dienstbenutzer — Wenn Sie den Aurora DSQL-Service für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die Anmeldeinformationen und Berechtigungen zur Verfügung, die Sie benötigen. Da Sie für Ihre Arbeit mehr Aurora DSQL-Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Wenn Sie in Aurora DSQL nicht auf eine Funktion zugreifen können, finden Sie weitere Informationen unterFehlerbehebung bei Amazon Aurora DSQL Identität und Zugriff.

Service-Administrator — Wenn Sie in Ihrem Unternehmen für die Aurora DSQL-Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf Aurora DSQL. Es ist Ihre Aufgabe, zu bestimmen, auf welche Funktionen und Ressourcen von Aurora DSQL Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anforderungen an Ihren IAM-Administrator senden, um die Berechtigungen der Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen darüber, wie Ihr Unternehmen IAM mit Aurora DSQL verwenden kann, finden Sie unter. So funktioniert Amazon Aurora DSQL mit IAM

IAM-Administrator — Wenn Sie ein IAM-Administrator sind, möchten Sie vielleicht mehr darüber erfahren, wie Sie Richtlinien schreiben können, um den Zugriff auf Aurora DSQL zu verwalten. Beispiele für identitätsbasierte Aurora DSQL-Richtlinien, die Sie in IAM verwenden können, finden Sie unter. Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS , übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter So melden Sie sich bei Ihrem an AWS-Konto im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert darauf zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, um Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode für die Selbstsignierung von Anforderungen finden Sie unter AWS Signature Version 4 für API-Anforderungen im IAM-Benutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen bereitstellen. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center - Benutzerhandbuch und AWS Multi-Faktor-Authentifizierung (MFA) in IAM im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen. Verwenden Sie diese nur, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen den Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, einem Web-Identitätsanbieter AWS Directory Service, dem Identity Center-Verzeichnis oder einem beliebigen Benutzer, der mithilfe AWS-Services von Anmeldeinformationen zugreift, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung

zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Eine IAM-Gruppe ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise eine Gruppe benennen IAMAdminsund dieser Gruppe Berechtigungen zur Verwaltung von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt. Sie ist einem IAM-Benutzer vergleichbar, jedoch nicht mit einer bestimmten Person verknüpft. Um vorübergehend eine IAM-Rolle in der zu übernehmen AWS Management Console, können Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Methoden für die Übernahme einer Rolle im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- Verbundbenutzerzugriff – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter Erstellen von Rollen für externe Identitätsanbieter (Verbund) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Temporäre IAM-Benutzerberechtigungen – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.
- Serviceübergreifender Zugriff — Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise in einem Service einen Anruf tätigen, ist es üblich, dass dieser Service Anwendungen in Amazon ausführt EC2 oder Objekte in Amazon S3 speichert. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward Access Sessions (FAS) — Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen an AWS-Service nachgelagerte Dienste zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle – Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM

erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

- Dienstbezogene Rolle — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

- Auf Amazon ausgeführte Anwendungen EC2 — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2 Instance ausgeführt werden und AWS API-Anfragen stellen AWS CLI . Dies ist dem Speichern von Zugriffsschlüsseln innerhalb der EC2 Instance vorzuziehen. Um einer EC2 Instanz eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instanzprofil, das an die Instanz angehängt ist. Ein Instanzprofil enthält die Rolle und ermöglicht Programmen, die auf der EC2 Instanz ausgeführt werden, temporäre Anmeldeinformationen abzurufen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Verwenden einer IAM-Rolle, um Berechtigungen für Anwendungen zu gewähren, die auf EC2 Amazon-Instances ausgeführt werden.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Die Berechtigungen in den Richtlinien legen fest, ob eine Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können AWS-Konto. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer Inline-Richtlinie wählen, finden Sie unter Auswählen zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter Übersicht über ACLs die Zugriffskontrollliste (ACL) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen – Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.
- Dienststeuerungsrichtlinien (SCPs) — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Objekte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie alle Funktionen in einer Organisation aktivieren, können Sie Richtlinien zur Servicesteuerung (SCPs) auf einige oder alle Ihre Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Root-Benutzer des AWS-Kontos Entitäten. Weitere Informationen zu Organizations und SCPs finden Sie unter Richtlinien zur Servicesteuerung im AWS Organizations Benutzerhandbuch.
- Ressourcenkontrollrichtlinien (RCPs) — RCPs sind JSON-Richtlinien, mit denen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten festlegen können, ohne die IAM-Richtlinien aktualisieren zu müssen, die jeder Ressource zugeordnet sind, deren Eigentümer Sie sind. Das RCP schränkt die Berechtigungen für Ressourcen in Mitgliedskonten ein und kann sich

auf die effektiven Berechtigungen für Identitäten auswirken, einschließlich der Root-Benutzer des AWS-Kontos, unabhängig davon, ob sie zu Ihrer Organisation gehören. Weitere Informationen zu Organizations RCPs, einschließlich einer Liste AWS-Services dieser Support-Leistungen RCPs, finden Sie unter Resource Control Policies (RCPs) im AWS Organizations Benutzerhandbuch.

- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAM-Benutzerhandbuch unter Bewertungslogik für Richtlinien.

So funktioniert Amazon Aurora DSQL mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf Aurora DSQL zu verwalten, sollten Sie sich darüber informieren, welche IAM-Funktionen mit Aurora DSQL verwendet werden können.

IAM-Funktionen, die Sie mit Amazon Aurora DSQL verwenden können

IAM-Feature	Aurora DSQL-Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Bedingungsschlüssel für die Richtlinie	Ja

IAM-Feature	Aurora DSQL-Unterstützung
ACLs	Nein
ABAC (Tags in Richtlinien)	Ja
Temporäre Anmeldeinformationen	Ja
Prinzipalberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Ja

Einen allgemeinen Überblick darüber, wie Aurora DSQL und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für Aurora DSQL

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Sie können den Prinzipal nicht in einer identitätsbasierten Richtlinie angeben, da er für den Benutzer oder die Rolle gilt, dem er zugeordnet ist. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für Aurora DSQL

Beispiele für identitätsbasierte Aurora DSQL-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Ressourcenbasierte Richtlinien in Aurora DSQL

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource unterscheiden AWS-Konten, muss ein IAM-Administrator des vertrauenswürdigen Kontos auch der Prinzipalentität (Benutzer oder Rolle) die Berechtigung zum Zugriff auf die Ressource erteilen. Sie erteilen Berechtigungen, indem Sie der juristischen Stelle eine identitätsbasierte Richtlinie anfügen. Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Politische Maßnahmen für Aurora DSQL

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API-Vorgang. Es gibt einige Ausnahmen, z. B. Aktionen, die nur mit Genehmigung durchgeführt werden können und für die es keinen passenden API-Vorgang

gibt. Es gibt auch einige Operationen, die mehrere Aktionen in einer Richtlinie erfordern. Diese zusätzlichen Aktionen werden als abhängige Aktionen bezeichnet.

Schließen Sie Aktionen in eine Richtlinie ein, um Berechtigungen zur Durchführung der zugeordneten Operation zu erteilen.

Eine Liste der Aurora DSQL-Aktionen finden Sie unter Von Amazon Aurora DSQL definierte Aktionen in der Service Authorization Reference.

Richtlinienaktionen in Aurora DSQL verwenden das folgende Präfix vor der Aktion:

```
dsql
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

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

Beispiele für identitätsbasierte Aurora DSQL-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Politische Ressourcen für Aurora DSQL

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Anweisungen müssen entweder ein – `Resource`oder ein `NotResource`-Element enthalten. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Sie können dies für Aktionen tun, die einen bestimmten Ressourcentyp unterstützen, der als Berechtigungen auf Ressourcenebene bezeichnet wird.

Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, z. B. Auflistungsoperationen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der Aurora DSQL-Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Von Amazon Aurora DSQL definierte Ressourcen in der Service Authorization Reference. Informationen darüber, mit welchen Aktionen Sie den ARN jeder Ressource angeben können, finden Sie unter Von Amazon Aurora DSQL definierte Aktionen.

Beispiele für identitätsbasierte Aurora DSQL-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Schlüssel für Richtlinienbedingungen für Aurora DSQL

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Condition` (oder `Condition` block) ermöglicht Ihnen die Angabe der Bedingungen, unter denen eine Anweisung wirksam ist. Das Element `Condition` ist optional. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt.

Wenn Sie mehrere `Condition`-Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen `Condition`-Element angeben, wertet AWS diese mittels einer logischen AND-Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, AWS wertet die Bedingung mithilfe einer logischen OR Operation aus. Alle Bedingungen müssen erfüllt werden, bevor die Berechtigungen der Anweisung gewährt werden.

Sie können auch Platzhaltervariablen verwenden, wenn Sie Bedingungen angeben. Beispielsweise können Sie einem IAM-Benutzer die Berechtigung für den Zugriff auf eine Ressource nur dann gewähren, wenn sie mit dessen IAM-Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie unter IAM-Richtlinienelemente: Variablen und Tags im IAM-Benutzerhandbuch.

AWS unterstützt globale Bedingungsschlüssel und dienstspezifische Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste der Aurora DSQL-Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für Amazon Aurora DSQL in der Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Von Amazon Aurora DSQL definierte Aktionen.

Beispiele für identitätsbasierte Aurora DSQL-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

ACLs in Aurora DSQL

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

ABAC mit Aurora DSQL

Unterstützt ABAC (Tags in Richtlinien): Ja

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen definiert werden. In AWS werden diese Attribute als Tags bezeichnet. Sie können Tags an IAM-Entitäten (Benutzer oder Rollen) und an viele AWS Ressourcen anhängen. Das Markieren von Entitäten und Ressourcen ist der erste Schritt von ABAC. Anschließend entwerfen Sie ABAC-Richtlinien, um Operationen zuzulassen, wenn das Tag des Prinzipals mit dem Tag der Ressource übereinstimmt, auf die sie zugreifen möchten.

ABAC ist in Umgebungen hilfreich, die schnell wachsen, und unterstützt Sie in Situationen, in denen die Richtlinienverwaltung mühsam wird.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Temporäre Anmeldeinformationen mit Aurora DSQL verwenden

Unterstützt temporäre Anmeldeinformationen: Ja

Einige funktionieren AWS-Services nicht, wenn Sie sich mit temporären Anmeldeinformationen anmelden. Weitere Informationen, einschließlich Informationen, die mit temporären Anmeldeinformationen AWS-Services funktionieren AWS-Services , finden Sie im IAM-Benutzerhandbuch unter Diese Option funktioniert mit IAM.

Sie verwenden temporäre Anmeldeinformationen, wenn Sie sich mit einer anderen AWS Management Console Methode als einem Benutzernamen und einem Passwort anmelden. Wenn Sie beispielsweise AWS über den Single Sign-On-Link (SSO) Ihres Unternehmens darauf zugreifen, werden bei diesem Vorgang automatisch temporäre Anmeldeinformationen erstellt. Sie erstellen auch automatisch temporäre Anmeldeinformationen, wenn Sie sich als Benutzer bei der Konsole anmelden und dann die Rollen wechseln. Weitere Informationen zum Wechseln von Rollen finden Sie unter Wechseln von einer Benutzerrolle zu einer IAM-Rolle (Konsole) im IAM-Benutzerhandbuch.

Mithilfe der AWS API AWS CLI oder können Sie temporäre Anmeldeinformationen manuell erstellen. Sie können diese temporären Anmeldeinformationen dann für den Zugriff verwenden AWS. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen in IAM.

Serviceübergreifende Prinzipalberechtigungen für Aurora DSQL

Unterstützt Forward Access Sessions (FAS): Ja

Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, kombiniert mit der Anforderung, Anfragen an nachgelagerte Dienste AWS-Service zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für Aurora DSQL

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Durch das Ändern der Berechtigungen für eine Servicerolle kann die Aurora DSQL-Funktionalität beeinträchtigt werden. Bearbeiten Sie Servicerollen nur, wenn Aurora DSQL Sie dazu anleitet.

Serviceverknüpfte Rollen für Aurora DSQL

Unterstützt serviceverknüpfte Rollen: Ja

Eine serviceverknüpfte Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Einzelheiten zum Erstellen oder Verwalten von serviceverknüpften Rollen für Aurora DSQL finden Sie unter. Verwenden von serviceverknüpften Rollen in Aurora DSQL

Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Standardmäßig sind Benutzer und Rollen nicht berechtigt, Aurora DSQL-Ressourcen zu erstellen oder zu ändern. Sie können auch keine Aufgaben mithilfe der AWS Management Console, AWS Command Line Interface (AWS CLI) oder AWS API ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu den von Aurora DSQL definierten Aktionen und Ressourcentypen, einschließlich des Formats ARNs für die einzelnen Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für Amazon Aurora DSQL in der Service Authorization Reference.

Themen

Bewährte Methoden für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand Aurora DSQL-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder diese löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Befolgen Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Anleitungen und Empfehlungen:

- Beginnen Sie mit AWS verwalteten Richtlinien und gehen Sie zu Berechtigungen mit den geringsten Rechten über — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter AWS -verwaltete Richtlinien oder AWS -verwaltete Richtlinien für Auftrags-Funktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und

Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. AWS CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der Aurora DSQL-Konsole

Um auf die Amazon Aurora DSQL-Konsole zugreifen zu können, benötigen Sie einen Mindestsatz an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den Aurora DSQL-Ressourcen in Ihrem AWS-Konto aufzulisten und anzuzeigen. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die Aurora DSQL-Konsole weiterhin verwenden können, fügen Sie den Entitäten auch die Aurora DSQL `AmazonAuroraDSQLConsoleFullAccess` - oder `AmazonAuroraDSQLReadOnlyAccess` AWS verwaltete Richtlinie hinzu. Weitere

Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie beinhaltet Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der OR-API. 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": "*"
        }
    ]
}
```

Fehlerbehebung bei Amazon Aurora DSQL Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit Aurora DSQL und IAM auftreten können.

Themen

Ich bin nicht berechtigt, eine Aktion in Aurora DSQL durchzuführen

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der `mateojackson` versucht, die Konsole zu verwenden, um Details zur *`my-dsql-cluster`* Ressource anzuzeigen, aber nicht über die *`GetCluster`* entsprechenden Berechtigungen verfügt.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der *`GetCluster`*-Aktion auf die *`my-dsql-cluster`*-Ressource zugreifen kann.

Wenden Sie sich an Ihren -Administrator, falls Sie weitere Unterstützung benötigen. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht berechtigt sind, die `iam:PassRole` Aktion durchzuführen, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an Aurora DSQL übergeben können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in Aurora DSQL auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine Aurora DSQL-Ressourcen ermöglichen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob Aurora DSQL diese Funktionen unterstützt, finden Sie unterSo funktioniert Amazon Aurora DSQL mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwenden von serviceverknüpften Rollen in Aurora DSQL

Aurora DSQL verwendet AWS Identity and Access Management (IAM) serviceverknüpfte Rollen. Eine serviceverknüpfte Rolle ist eine einzigartige Art von IAM-Rolle, die direkt mit Aurora DSQL verknüpft ist. Service-verknüpfte Rollen sind von Aurora DSQL vordefiniert und beinhalten alle Berechtigungen, die der Service benötigt, um im Namen Ihres Aurora DSQL-Clusters aufzurufen AWS-Services .

Serviceverknüpfte Rollen erleichtern den Einrichtungsprozess, da Sie die erforderlichen Berechtigungen für die Verwendung von Aurora DSQL nicht manuell hinzufügen müssen. Wenn Sie einen Cluster erstellen, erstellt Aurora DSQL automatisch eine serviceverknüpfte Rolle für Sie. Sie können die serviceverknüpfte Rolle erst löschen, nachdem Sie alle Ihre Cluster gelöscht haben. Dies schützt Ihre Aurora DSQL-Ressourcen, da Sie nicht versehentlich die für den Zugriff auf die Ressourcen erforderlichen Berechtigungen entfernen können.

Informationen zu anderen Services, die serviceverknüpfte Rollen unterstützen, finden Sie unter AWS-Services die mit IAM arbeiten. Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rolle angegeben ist. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

Servicebezogene Rollen sind in allen unterstützten Aurora DSQL-Regionen verfügbar.

Dienstbezogene Rollenberechtigungen für Aurora DSQL

Aurora DSQL verwendet die serviceverknüpfte Rolle mit dem Namen `AWSServiceRoleForAuroraDsql` — Erlaubt Amazon Aurora DSQL, AWS Ressourcen in Ihrem Namen zu erstellen und zu verwalten. Diese verwaltete Richtlinie ist mit der folgenden serviceverknüpften Rolle verbunden: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Möglicherweise wird die folgende Fehlermeldung angezeigt:. `You don't have the permissions to create an Amazon Aurora DSQL service-linked role` Wenn Sie diesen Fehler erhalten, überprüfen Sie, ob die folgenden Berechtigungen aktiviert sind:
>
> JSON
>
> ```
> {
> ```

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

Weitere Informationen finden Sie unter Dienstbezogene Rollenberechtigungen.

Erstellen einer serviceverknüpften Rolle

Sie müssen keine mit dem DSQLService LinkedRolePolicy Service verknüpfte Aurora-Rolle manuell erstellen. Aurora DSQL erstellt die serviceverknüpfte Rolle für Sie. Wenn die mit dem DSQLService LinkedRolePolicy Aurora-Dienst verknüpfte Rolle aus Ihrem Konto gelöscht wurde, erstellt Aurora DSQL die Rolle, wenn Sie einen neuen Aurora DSQL-Cluster erstellen.

Bearbeiten einer serviceverknüpften Rolle

Aurora DSQL erlaubt es Ihnen nicht, die mit dem DSQLService LinkedRolePolicy Aurora-Dienst verknüpfte Rolle zu bearbeiten. Nachdem Sie eine serviceverknüpfte Rolle erstellt haben, können Sie den Namen der Rolle nicht mehr ändern, da verschiedene Entitäten auf die Rolle verweisen könnten. Sie können die Beschreibung der Rolle jedoch mithilfe der IAM-Konsole, der AWS Command Line Interface (AWS CLI) oder der IAM-API bearbeiten.

Löschen Sie eine serviceverknüpfte Rolle

Wenn Sie ein Feature oder einen Service, die bzw. der eine servicegebundene Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise verfügen Sie nicht über eine ungenutzte Entität, die nicht aktiv überwacht oder verwaltet wird.

Bevor Sie eine dienstverknüpfte Rolle für ein Konto löschen können, müssen Sie alle Cluster im Konto löschen.

Sie können die IAM-Konsole, die oder die IAM-API verwenden AWS CLI, um eine dienstverknüpfte Rolle zu löschen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erstellen einer serviceverknüpften Rolle.

Unterstützte Regionen für serviceverknüpfte Aurora-DSQL-Rollen

Aurora DSQL unterstützt die Verwendung von serviceverknüpften Rollen in allen Regionen, in denen der Service verfügbar ist. Weitere Informationen finden Sie unter AWS Regionen und Endpunkte.

Verwenden von IAM-Bedingungsschlüsseln mit Amazon Aurora DSQL

Wenn Sie in Aurora DSQL Berechtigungen gewähren, können Sie Bedingungen angeben, die bestimmen, wie eine Berechtigungsrichtlinie wirksam wird. Im Folgenden finden Sie Beispiele dafür, wie Sie Bedingungsschlüssel in Aurora DSQL-Berechtigungsrichtlinien verwenden können.

Beispiel 1: Erteilen Sie die Erlaubnis, einen Cluster in einem bestimmten AWS-Region

Die folgende Richtlinie erteilt die Genehmigung zur Erstellung von Clustern in den Regionen USA Ost (Nord-Virginia) und USA Ost (Ohio). Diese Richtlinie verwendet den Ressourcen-ARN, um die zulässigen Regionen zu begrenzen, sodass Aurora DSQL nur Cluster erstellen kann, wenn dieser ARN im `Resource` Abschnitt der Richtlinie angegeben ist.

JSON

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

```
}
```

Beispiel 2: Erteilen Sie die Erlaubnis, einen Cluster mit mehreren Regionen in bestimmten s zu erstellen AWS-Region

Die folgende Richtlinie erteilt die Genehmigung zur Erstellung von Clustern mit mehreren Regionen in den Regionen USA Ost (Nord-Virginia) und USA Ost (Ohio). Diese Richtlinie verwendet den Ressourcen-ARN, um die zulässigen Regionen zu begrenzen, sodass Aurora DSQL Cluster mit mehreren Regionen nur erstellen kann, wenn dieser ARN im `Resource` Abschnitt der Richtlinie angegeben ist. Beachten Sie, dass für die Erstellung von Clustern mit mehreren Regionen auch die `AddPeerCluster` Berechtigungen `PutMultiRegionPropertiesPutWitnessRegion`, und in jeder angegebenen Region erforderlich sind.

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/*"
        ]
      }
    ]
}
```

Beispiel 3: Erteilen Sie die Berechtigung zum Erstellen eines Clusters mit mehreren Regionen mit einer bestimmten Zeugenregion

Die folgende Richtlinie verwendet einen Aurora `dsql:WitnessRegion` DSQL-Bedingungsschlüssel und ermöglicht es einem Benutzer, Cluster mit mehreren Regionen mit einer Zeugenregion in USA West (Oregon) zu erstellen. Wenn Sie die `dsql:WitnessRegion` Bedingung nicht angeben, können Sie eine beliebige Region als Zeugenregion verwenden.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": [
                        "us-west-2"
                    ]
                }
            }
        }
    ]
}
```

Reaktion auf Vorfälle in Amazon Aurora DSQL

Sicherheit hat bei AWS uns höchste Priorität. AWS Verwaltet im Rahmen des Modells der gemeinsamen Verantwortung in der AWS Cloud eine Rechenzentrums-, Netzwerk- und Softwarearchitektur, die die Anforderungen der sicherheitssensibelsten Unternehmen erfüllt. AWS ist für jegliche Reaktion auf Vorfälle in Bezug auf den Amazon Aurora DSQL-Service selbst verantwortlich. Außerdem tragen Sie als AWS Kunde gemeinsam die Verantwortung für die Aufrechterhaltung der Sicherheit in der Cloud. Das bedeutet, dass Sie die Sicherheit, die Sie implementieren möchten, anhand der AWS Tools und Funktionen, auf die Sie Zugriff haben, kontrollieren. Darüber hinaus sind Sie im Rahmen des Modells der gemeinsamen Verantwortung für die Reaktion auf Vorfälle verantwortlich.

Indem Sie eine Sicherheitsbasis einrichten, die den Zielen Ihrer in der Cloud ausgeführten Anwendungen entspricht, können Sie Abweichungen erkennen, auf die Sie reagieren können. Damit Sie besser verstehen, welche Auswirkungen die Reaktion auf Vorfälle und Ihre Entscheidungen auf Ihre Unternehmensziele haben, empfehlen wir Ihnen, sich die folgenden Ressourcen anzusehen:

- AWS Leitfaden zur Reaktion auf Sicherheitsvorfälle
- AWS Bewährte Methoden für Sicherheit, Identität und Compliance
- Das Whitepaper AWS Cloud Adoption Framework (CAF) aus der Sicherheitsperspektive

Amazon GuardDuty ist ein verwalteter Service zur Bedrohungserkennung, der kontinuierlich bösartiges oder unbefugtes Verhalten überwacht, um Kunden dabei zu unterstützen, ihre Workloads zu schützen AWS-Konten und verdächtige Aktivitäten zu identifizieren, bevor sie zu einem Vorfall eskalieren. Er überwacht Aktivitäten wie ungewöhnliche API-Aufrufe oder potenziell unautorisierte Bereitstellungen, was auf eine mögliche Kompromittierung von Konten oder Ressourcen oder die Entdeckung durch böswillige Akteure hindeutet. Amazon GuardDuty ist beispielsweise in der Lage, verdächtige Aktivitäten in Amazon Aurora DSQL zu erkennen APIs, z. B. wenn sich ein Benutzer von einem neuen Standort aus anmeldet und einen neuen Cluster erstellt.

Konformitätsvalidierung für Amazon Aurora DSQL

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Compliance und Governance im Bereich Sicherheit – In diesen Anleitungen für die Lösungsimplementierung werden Überlegungen zur Architektur behandelt. Außerdem werden Schritte für die Bereitstellung von Sicherheits- und Compliance-Features beschrieben.
- Referenz für berechtigte HIPAA-Services – Listet berechtigte HIPAA-Services auf. Nicht alle AWS-Services sind HIPAA-fähig.
- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- AWS Leitfäden zur Einhaltung von Vorschriften für Kunden — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS-Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- Evaluierung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Auf diese AWS-Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Die Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerelementreferenz.
- Amazon GuardDuty — Dies AWS-Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.

- AWS Audit Manager— Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Resilienz in Amazon Aurora DSQL

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones (AZ). AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren. Aurora DSQL wurde so konzipiert, dass Sie die Vorteile der AWS regionalen Infrastruktur nutzen und gleichzeitig die höchste Datenbankverfügbarkeit bieten können. Standardmäßig verfügen Cluster mit einer Region in Aurora DSQL über Multi-AZ-Verfügbarkeit, sodass größere Komponentenausfälle und Infrastrukturunterbrechungen, die den Zugriff auf eine vollständige AZ beeinträchtigen könnten, toleriert werden. Cluster mit mehreren Regionen bieten alle Vorteile der Multi-AZ-Resilienz und bieten gleichzeitig eine äußerst konsistente Datenbankverfügbarkeit, selbst in Fällen, in denen Anwendungsclients nicht darauf zugreifen können. AWS-Region

Weitere Informationen zu Availability Zones AWS-Regionen und Availability Zones finden Sie unter Globale Infrastruktur.AWS

Zusätzlich zur AWS globalen Infrastruktur bietet Aurora DSQL mehrere Funktionen, um Ihre Datenstabilität und Backup-Anforderungen zu erfüllen.

Backup und Backup

Aurora DSQL unterstützt Backup und Wiederherstellung mit AWS-Backup-Konsole. Sie können eine vollständige Sicherung und Wiederherstellung für Ihre Cluster mit einer oder mehreren Regionen durchführen. Weitere Informationen finden Sie unter Backup und Wiederherstellung für Amazon Aurora DSQL.

Replikation

Standardmäßig schreibt Aurora DSQL alle Schreibtransaktionen in ein verteiltes Transaktionslog ein und repliziert alle übergebenen Protokolldaten synchron in drei Benutzerspeicherreplikaten. AZs

Cluster mit mehreren Regionen bieten umfassende regionsübergreifende Replikationsfunktionen zwischen Lese- und Schreibregionen.

Eine ausgewiesene Zeugenregion unterstützt ausschließlich Transaktionsprotokoll-Schreibvorgänge und verbraucht keinen Speicherplatz. Zeugenregionen haben keinen Endpunkt. Das bedeutet, dass Zeugenregionen nur verschlüsselte Transaktionsprotokolle speichern, keine Verwaltung oder Konfiguration erfordern und für Benutzer nicht zugänglich sind.

Aurora DSQL-Transaktionsprotokolle und Benutzerspeicher werden so verteilt, dass alle Daten den Aurora DSQL-Abfrageprozessoren als ein einziges logisches Volume präsentiert werden. Aurora DSQL teilt, zusammenführt und repliziert Daten automatisch auf der Grundlage des Primärschlüsselbereichs der Datenbank und der Zugriffsmuster. Aurora DSQL skaliert Lesereplikate automatisch, sowohl nach oben als auch nach unten, basierend auf der Lesezugriffshäufigkeit.

Cluster-Speicherreplikate sind auf eine Speicherflotte mit mehreren Mandanten verteilt. Wenn eine Komponente oder AZ beeinträchtigt wird, leitet Aurora DSQL den Zugriff automatisch auf überlebende Komponenten um und repariert asynchron fehlende Replikate. Sobald Aurora DSQL die beeinträchtigten Replikate repariert hat, fügt Aurora DSQL sie automatisch wieder dem Speicherquorum hinzu und stellt sie Ihrem Cluster zur Verfügung.

Hohe Verfügbarkeit

Standardmäßig sind Cluster mit einer Region und mehreren Regionen in Aurora DSQL aktiv-aktiv, und Sie müssen keine Cluster manuell bereitstellen, konfigurieren oder neu konfigurieren. Aurora DSQL automatisiert die Cluster-Wiederherstellung vollständig, wodurch herkömmliche primär-sekundäre Failover-Operationen überflüssig werden. Die Replikation erfolgt immer synchron und mehrfach AZs, sodass kein Risiko eines Datenverlusts aufgrund von Verzögerungen bei der Replikation oder eines Failovers auf eine asynchrone sekundäre Datenbank während der Wiederherstellung nach einem Ausfall besteht.

Cluster mit einer Region bieten einen redundanten Multi-AZ-Endpunkt, der automatisch den gleichzeitigen Zugriff mit hoher Datenkonsistenz über drei Bereiche hinweg ermöglicht. AZs Das bedeutet, dass Benutzerspeicherreplikate auf einem dieser drei Geräte AZs immer dasselbe Ergebnis an einen oder mehrere Leser zurückgeben und immer für Schreibvorgänge verfügbar sind. Diese starke Konsistenz und Multi-AZ-Resilienz ist in allen Regionen für Aurora DSQL-Cluster mit mehreren Regionen verfügbar. Das bedeutet, dass Cluster mit mehreren Regionen zwei stark konsistente regionale Endpunkte bieten, sodass Clients wahllos in einer der beiden Regionen lesen oder schreiben können, ohne dass die Replikationsverzögerung beim Commit auftritt.

Aurora DSQL bietet eine Verfügbarkeit von 99,99% für Cluster mit einer Region und 99,999% für Cluster mit mehreren Regionen.

Infrastruktursicherheit in Amazon Aurora DSQL

Als verwalteter Service ist Amazon Aurora DSQL durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die in Best Practices for Security, Identity and Compliance beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf Aurora DSQL zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Clients müssen außerdem Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Verwaltung und Verbindung zu Amazon Aurora DSQL-Clustern mithilfe von AWS PrivateLink

Mit AWS PrivateLink for Amazon Aurora DSQL können Sie Amazon VPC-Schnittstellenendpunkte (Schnittstellenendpunkte) in Ihrer Amazon Virtual Private Cloud bereitstellen. Auf diese Endpunkte kann direkt von Anwendungen aus zugegriffen werden, die sich vor Ort befinden AWS Direct Connect, über Amazon VPC und/oder auf andere Weise AWS-Region über Amazon VPC Peering. Mithilfe von Endpunkten AWS PrivateLink und Schnittstellen können Sie die private Netzwerkkonnektivität zwischen Ihren Anwendungen und Aurora DSQL vereinfachen.

Anwendungen in Ihrer Amazon VPC können über Amazon VPC-Schnittstellenendpunkte auf Aurora DSQL zugreifen, ohne dass öffentliche IP-Adressen erforderlich sind.

Schnittstellenendpunkte werden durch eine oder mehrere elastische Netzwerkschnittstellen (ENIs) repräsentiert, denen private IP-Adressen aus Subnetzen in Ihrer Amazon VPC zugewiesen wurden. Anfragen an Aurora DSQL über Schnittstellenendpunkte bleiben im AWS Netzwerk. Weitere Informationen darüber, wie Sie Ihre Amazon VPC mit Ihrem lokalen Netzwerk verbinden, finden Sie im AWS Direct Connect Benutzerhandbuch und im AWS Site-to-Site VPN VPN-Benutzerhandbuch.

Allgemeine Informationen zu Schnittstellenendpunkten finden Sie unter Zugreifen auf einen AWS Service mithilfe eines Amazon VPC-Schnittstellenendpunkts im AWS PrivateLinkBenutzerhandbuch.

Arten von Amazon VPC-Endpunkten für Aurora DSQL

Aurora DSQL erfordert zwei verschiedene Arten von AWS PrivateLink Endpunkten.

1. Verwaltungsendpunkt — Dieser Endpunkt wird für Verwaltungsvorgänge wie,`get`, `create` `update``delete`, und `list` auf Aurora SQL-Clustern verwendet. Siehe Verwaltung von Aurora DSQL-Clustern mit AWS PrivateLink.
2. Verbindungsendpunkt — Dieser Endpunkt wird für die Verbindung zu Aurora DSQL-Clustern über PostgreSQL-Clients verwendet. Siehe Verbindung zu Aurora DSQL-Clustern herstellen mit AWS PrivateLink.

Überlegungen bei der Verwendung von AWS PrivateLink Aurora DSQL

Überlegungen zu Amazon VPC gelten AWS PrivateLink für Aurora DSQL. Weitere Informationen finden Sie im AWS PrivateLink Handbuch unter Zugreifen auf einen AWS Dienst über eine Schnittstelle, VPC-Endpunkt und AWS PrivateLink Kontingente.

Verwaltung von Aurora DSQL-Clustern mit AWS PrivateLink

Sie können die AWS Command Line Interface oder AWS Software Development Kits (SDKs) verwenden, um Aurora DSQL-Cluster über Aurora DSQL-Schnittstellenendpunkte zu verwalten.

Erstellen eines Amazon VPC-Endpunkts

Informationen zum Erstellen eines Amazon VPC-Schnittstellenendpunkts finden Sie unter Erstellen eines Amazon VPC-Endpunkts im AWS PrivateLink Handbuch.

```
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 \
```

Um den standardmäßigen regionalen DNS-Namen für Aurora DSQL-API-Anfragen zu verwenden, deaktivieren Sie privates DNS nicht, wenn Sie den Aurora DSQL-Schnittstellenendpunkt erstellen.

Wenn privates DNS aktiviert ist, werden Anfragen an den Aurora DSQL-Service, die von Ihrer Amazon VPC aus gestellt werden, automatisch auf die private IP-Adresse des Amazon VPC-Endpunkts und nicht auf den öffentlichen DNS-Namen aufgelöst. Wenn privates DNS aktiviert ist, werden Aurora DSQL-Anfragen, die in Ihrer Amazon VPC gestellt werden, automatisch auf Ihren Amazon VPC-Endpunkt aufgelöst.

Wenn privates DNS nicht aktiviert ist, verwenden Sie die `--endpoint-url` Parameter `--region` und mit AWS CLI Befehlen, um Aurora DSQL-Cluster über Aurora DSQL-Schnittstellenendpunkte zu verwalten.

Cluster mithilfe einer Endpunkt-URL auflisten

Ersetzen Sie im folgenden Beispiel den AWS-Region `us-east-1` und den DNS-Namen der Amazon VPC-Endpunkt-ID `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-
east-1.vpce.amazonaws.com list-clusters
```

API-Operationen

Dokumentation zur Verwaltung von Ressourcen in Aurora DSQL finden Sie in der Aurora DSQL API-Referenz.

Verwaltung von Endpunktrichtlinien

Durch gründliches Testen und Konfigurieren der Amazon VPC-Endpunktrichtlinien können Sie sicherstellen, dass Ihr Aurora DSQL-Cluster sicher und konform ist und den spezifischen Zugriffskontroll- und Governance-Anforderungen Ihres Unternehmens entspricht.

Beispiel: Vollständige Aurora DSQL-Zugriffsrichtlinie

Die folgende Richtlinie gewährt vollen Zugriff auf alle Aurora DSQL-Aktionen und -Ressourcen über den angegebenen Amazon VPC-Endpunkt.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
```

```
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": "*",
        "Action": "dsql:*",
        "Resource": "*"
      }
    ]
  }'
```

Beispiel: Eingeschränkte Aurora DSQL-Zugriffsrichtlinie

Die folgende Richtlinie erlaubt nur diese Aurora DSQL-Aktionen.

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

Alle anderen Aurora DSQL-Aktionen werden verweigert.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Verbindung zu Aurora DSQL-Clustern herstellen mit AWS PrivateLink

Sobald Ihr AWS PrivateLink Endpunkt eingerichtet und aktiv ist, können Sie mit einem PostgreSQL-Client eine Verbindung zu Ihrem Aurora DSQL-Cluster herstellen. In den folgenden Verbindungsanweisungen werden die Schritte zur Erstellung des richtigen Hostnamens für die Verbindung über den Endpunkt beschrieben. AWS PrivateLink

Einen AWS PrivateLink Verbindungsendpunkt einrichten

Schritt 1: Holen Sie sich den Dienstnamen für Ihren Cluster

Wenn Sie einen AWS PrivateLink Endpunkt für die Verbindung zu Ihrem Cluster erstellen, müssen Sie zuerst den clusterspezifischen Dienstnamen abrufen.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Beispielantwort

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

Der Dienstname enthält einen Bezeichner, wie `dsql-fnh4` im Beispiel. Diese Kennung wird auch benötigt, wenn der Hostname für die Verbindung zu Ihrem Cluster erstellt wird.

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);
```

Schritt 2: Amazon VPC-Endpunkt erstellen

Erstellen Sie mit dem im vorherigen Schritt erhaltenen Servicenamen einen Amazon VPC-Endpunkt.

> **Important**
>
> Die folgenden Verbindungsanweisungen funktionieren nur für Verbindungen zu Clustern, wenn Private DNS aktiviert ist. Verwenden Sie das `--no-private-dns-enabled` Flag nicht, wenn Sie den Endpunkt erstellen, da dadurch die unten aufgeführten Verbindungsanweisungen nicht ordnungsgemäß funktionieren. Wenn Sie privates DNS deaktivieren, müssen Sie Ihren eigenen privaten DNS-Wildcard-Eintrag erstellen, der auf den erstellten Endpunkt verweist.

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
```

Beispielantwort

```
{
    "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

Verwenden Sie eine Endpunkt-URL für 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);
```

Über einen Verbindungsendpunkt eine Verbindung zu einem Aurora DSQL-Cluster herstellen AWS PrivateLink

Sobald Ihr AWS PrivateLink Endpunkt eingerichtet und aktiv ist (stellen Sie sicher, dass dies der Fall `State` ist`available`), können Sie über einen PostgreSQL-Client eine Verbindung zu Ihrem Aurora DSQL-Cluster herstellen. Anweisungen zur Verwendung von finden Sie in den Anleitungen unter Programmieren mit Aurora DSQL. AWS SDKs Sie müssen den Cluster-Endpunkt so ändern, dass er dem Hostnamenformat entspricht.

Den Hostnamen konstruieren

Der Hostname für die Verbindung AWS PrivateLink unterscheidet sich vom öffentlichen DNS-Hostnamen. Sie müssen ihn mit den folgenden Komponenten erstellen.

1. `Your-cluster-id`
2. Die Dienst-ID aus dem Dienstnamen. Beispiel: `dsql-fnh4`
3. Der AWS-Region

Verwenden Sie das folgende Format: *`cluster-id`*`.`*`service-identifier`*`.`*`region`*`.on.aws`

Beispiel: Verbindung mit 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
```

Beheben von Problemen mit AWS PrivateLink

Häufige Probleme und Lösungen

In der folgenden Tabelle sind häufig auftretende Probleme und Lösungen im Zusammenhang AWS PrivateLink mit Aurora DSQL aufgeführt.

Problem	Mögliche Ursache	Lösung
Verbindungstimeout	Sicherheitsgruppe nicht richtig konfiguriert	Verwenden Sie Amazon VPC Reachability Analyzer, um sicherzustellen, dass Ihr Netzwerk-Setup Datenverkehr auf Port 5432 zulässt.
Fehler bei der DNS-Auflösung	Privates DNS ist nicht aktiviert	Stellen Sie sicher, dass der Amazon VPC-Endpunkt mit aktiviertem privaten DNS erstellt wurde.
Fehler bei der Authentifizierung	Falsche Anmeldeinformationen oder abgelaufenes Token	Generieren Sie ein neues Authentifizierungstoken und überprüfen Sie den Benutzernamen.
Der Dienstname wurde nicht gefunden	Falsche Cluster-ID	Überprüfen Sie Ihre Cluster-ID und AWS-Region beim Abrufen des Dienstnamens noch einmal.

Verwandte Ressourcen

Weitere Informationen finden Sie in den folgenden Ressourcen:

- Amazon Aurora DSQL-Benutzerhandbuch
- AWS PrivateLink -Dokumentation
- Greifen Sie auf AWS Dienste zu über AWS PrivateLink

Konfiguration und Schwachstellenanalyse in Amazon Aurora DSQL

AWS kümmert sich um grundlegende Sicherheitsaufgaben wie das Patchen von Gastbetriebssystemen (OS) und Datenbanken, die Firewall-Konfiguration und die Notfallwiederherstellung. Diese Verfahren wurden von qualifizierten Dritten überprüft und zertifiziert. Weitere Informationen finden Sie in den folgenden Ressourcen:

- Modell der geteilten Verantwortung
- Amazon Web Services: Übersicht über Sicherheitsverfahren (Whitepaper)

Serviceübergreifende Confused-Deputy-Prävention

Das Confused-Deputy-Problem ist ein Sicherheitsproblem, bei dem eine juristische Stelle, die nicht über die Berechtigung zum Ausführen einer Aktion verfügt, eine privilegiertere juristische Stelle zwingen kann, die Aktion auszuführen. In AWS kann ein dienstübergreifendes Identitätswechsels zum Problem des verwirrten Stellvertreters führen. Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der aufrufende Service kann manipuliert werden, um seine Berechtigungen zu verwenden, um Aktionen auf die Ressourcen eines anderen Kunden auszuführen, für die er sonst keine Zugriffsberechtigung haben sollte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Serviceprinzipalen schützen können, die Zugriff auf Ressourcen in Ihrem Konto erhalten haben.

Wir empfehlen, die Kontextschlüssel `aws:SourceArn`und die `aws:SourceAccount`globalen Bedingungsschlüssel in Ressourcenrichtlinien zu verwenden, um die Berechtigungen einzuschränken, die Amazon Aurora DSQL einem anderen Service für die Ressource gewährt. Verwenden Sie `aws:SourceArn`, wenn Sie nur eine Ressource mit dem betriebsübergreifenden Zugriff verknüpfen möchten. Verwenden Sie `aws:SourceAccount`, wenn Sie zulassen möchten, dass Ressourcen in diesem Konto mit der betriebsübergreifenden Verwendung verknüpft werden.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontext-Schlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Wenn Sie den vollständigen ARN der Ressource nicht kennen oder wenn Sie mehrere Ressourcen angeben, verwenden Sie den globalen Kontextbedingungsschlüssel `aws:SourceArn` mit Platzhalterzeichen (*) für die unbekannten Teile des ARN. Beispiel, `arn:aws:dsql:*:123456789012:*`.

Wenn der `aws:SourceArn`-Wert die Konto-ID nicht enthält, z. B. einen Amazon-S3-Bucket-ARN, müssen Sie beide globale Bedingungskontextschlüssel verwenden, um Berechtigungen einzuschränken.

Der `aws:SourceArn`-Wert muss ResourceDescription lauten.

Das folgende Beispiel zeigt, wie Sie die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Aurora DSQL verwenden können, um das Problem des verwirrten Stellvertreters zu verhindern.

JSON

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

Bewährte Sicherheitsmethoden für Aurora DSQL

Aurora DSQL bietet eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

Themen

Bewährte Methoden zur Detektivsicherheit für Aurora DSQL

Zusätzlich zu den folgenden Möglichkeiten zur sicheren Verwendung von Aurora DSQL finden Sie unter Sicherheit in AWS Well-Architected Tool mehr darüber, wie Cloud-Technologien Ihre Sicherheit verbessern.

CloudWatch Amazon-Alarme

Mithilfe von CloudWatch Amazon-Alarmen beobachten Sie eine einzelne Metrik über einen von Ihnen angegebenen Zeitraum. Wenn die Metrik einen bestimmten Schwellenwert überschreitet, wird eine Benachrichtigung an ein Amazon SNS SNS-Thema oder eine AWS Auto Scaling Richtlinie gesendet. CloudWatch Alarme lösen keine Aktionen aus, da sie sich in einem bestimmten Status befinden. Der Status muss sich stattdessen geändert haben und für eine festgelegte Anzahl an Zeiträumen aufrechterhalten worden sein.

Kennzeichnen Sie Ihre Aurora DSQL-Ressourcen zur Identifizierung und Automatisierung

Sie können Ihren AWS Ressourcen Metadaten in Form von Tags zuweisen. Jedes Tag ist eine einfache Bezeichnung, die aus einem benutzerdefinierten Schlüssel und einem optionalen Wert besteht, der das Verwalten, Suchen und Filtern von Ressourcen erleichtern kann.

Tagging ermöglicht die Implementierung gruppierter Steuerelemente. Obwohl es keine inhärenten Typen von Tags gibt, können Sie Ressourcen nach Zweck, Besitzer, Umgebung oder anderen Kriterien kategorisieren. Im Folgenden sind einige Beispiele aufgeführt:

- Sicherheit – Wird verwendet, um Anforderungen wie Verschlüsselung zu bestimmen.

- Vertraulichkeit – Eine Kennung für die spezifische Datenvertraulichkeitsebene, die eine Ressource unterstützt
- Umgebung – Wird verwendet, um zwischen Entwicklungs-, Test- und Produktionsinfrastruktur zu unterscheiden.

Sie können Ihren AWS Ressourcen Metadaten in Form von Tags zuweisen. Jedes Tag ist eine einfache Bezeichnung, die aus einem benutzerdefinierten Schlüssel und einem optionalen Wert besteht, der das Verwalten, Suchen und Filtern von Ressourcen erleichtern kann.

Tagging ermöglicht die Implementierung gruppierter Steuerelemente. Obwohl es keine inhärenten Typen von Tags gibt, können Sie -Ressourcen nach Zweck, Eigentümer, Umgebung oder anderen Kriterien kategorisieren. Im Folgenden sind einige Beispiele aufgeführt:

- Sicherheit — wird verwendet, um Anforderungen wie Verschlüsselung zu ermitteln.
- Vertraulichkeit — eine Kennung für die spezifische Datenvertraulichkeitsstufe, die eine Ressource unterstützt.
- Umgebung — wird verwendet, um zwischen Entwicklungs-, Test- und Produktionsinfrastruktur zu unterscheiden.

Weitere Informationen finden Sie unter Bewährte Methoden zum Kennzeichnen von AWS Ressourcen.

Bewährte Methoden zur präventiven Sicherheit für Aurora DSQL

Zusätzlich zu den folgenden Möglichkeiten zur sicheren Verwendung von Aurora DSQL finden Sie unter Sicherheit in AWS Well-Architected Tool mehr darüber, wie Cloud-Technologien Ihre Sicherheit verbessern.

Verwenden Sie IAM-Rollen, um den Zugriff auf Aurora DSQL zu authentifizieren.

Benutzer, Anwendungen und andere, AWS-Services die auf Aurora DSQL zugreifen, müssen gültige AWS Anmeldeinformationen in der AWS API und in den AWS CLI Anfragen angeben. Sie sollten AWS Anmeldeinformationen nicht direkt in der Anwendung oder den EC2 Instanzen speichern. Dabei handelt es sich um langfristige Anmeldeinformationen, die nicht automatisch rotiert werden. Wenn diese Anmeldeinformationen kompromittiert werden, hat dies erhebliche Auswirkungen auf das Geschäft. Mit einer IAM-Rolle können Sie temporäre Zugriffsschlüssel abrufen, mit denen Sie auf Ressourcen zugreifen AWS-Services können.

Weitere Informationen finden Sie unter Authentifizierung und Autorisierung für Aurora DSQL.

Verwenden Sie IAM-Richtlinien für die Aurora DSQL-Basisautorisierung.

Wenn Sie Berechtigungen gewähren, entscheiden Sie, wer sie erhält, für welche Aurora DSQL-API-Operationen sie Berechtigungen erhalten und welche spezifischen Aktionen Sie für diese Ressourcen zulassen möchten. Die Implementierung der geringsten Rechte ist der Schlüssel zur Verringerung des Sicherheitsrisikos und der Auswirkungen, die sich aus Fehlern oder böswilligen Absichten ergeben können.

Hängen Sie Berechtigungsrichtlinien an IAM-Rollen an und gewähren Sie Berechtigungen zur Ausführung von Vorgängen auf Aurora DSQL-Ressourcen. Ebenfalls verfügbar sind Berechtigungsgrenzen für IAM-Entitäten, mit denen Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität gewähren kann.

Ähnlich wie bei den bewährten Methoden für Root-Benutzer AWS-Konto sollten Sie die `admin` Rolle in Aurora DSQL nicht für alltägliche Operationen verwenden. Stattdessen empfehlen wir Ihnen, benutzerdefinierte Datenbankrollen zu erstellen, um Ihren Cluster zu verwalten und eine Verbindung zu ihm herzustellen. Weitere Informationen finden Sie unter Zugreifen auf Aurora DSQL und Grundlegendes zur Authentifizierung und Autorisierung für Aurora DSQL.

Einsatz **`verify-full`** in Produktionsumgebungen.

Mit dieser Einstellung wird überprüft, ob das Serverzertifikat von einer vertrauenswürdigen Zertifizierungsstelle signiert ist und ob der Serverhostname mit dem Zertifikat übereinstimmt.

Aktualisieren Sie Ihren PostgreSQL-Client

Aktualisieren Sie Ihren PostgreSQL-Client regelmäßig auf die neueste Version, um von Sicherheitsverbesserungen zu profitieren. Wir empfehlen die Verwendung von PostgreSQL Version 17.

Taggen von Ressourcen in Aurora DSQL

In AWS sind Tags benutzerdefinierte Schlüssel-Wert-Paare, die Sie definieren und mit Aurora DSQL-Ressourcen wie Clustern verknüpfen. Tags sind optional. Wenn Sie einen Schlüssel angeben, ist der Wert optional.

Sie können die AWS Management Console, oder die verwenden AWS CLI, AWS SDKs um Tags auf Aurora DSQL-Clustern hinzuzufügen, aufzulisten und zu löschen. Sie können Tags während und nach der Clustererstellung mithilfe der AWS Konsole hinzufügen. AWS CLI Verwenden Sie die `TagResource` Operation, um einen Cluster nach der Erstellung mit dem Tag zu versehen.

Cluster mit einem Namen kennzeichnen

Aurora DSQL erstellt Cluster mit einer global eindeutigen Kennung, die als Amazon Resource Name (ARN) zugewiesen wird. Wenn Sie Ihrem Cluster einen benutzerfreundlichen Namen zuweisen möchten, empfehlen wir Ihnen, ein Tag zu verwenden.

Wenn Sie eine Konsole mit der Aurora DSQL-Konsole erstellen, erstellt Aurora DSQL automatisch ein Tag. Dieses Tag hat den Schlüssel Name und einen automatisch generierten Wert, der den Namen des Clusters darstellt. Dieser Wert ist konfigurierbar, sodass Sie Ihrem Cluster einen benutzerfreundlicheren Namen zuweisen können. Wenn ein Cluster über ein Name-Tag mit einem zugehörigen Wert verfügt, können Sie den Wert in der gesamten Aurora DSQL-Konsole sehen.

Anforderungen zum Markieren

Für Tags gelten zwei Anforderungen:

- Schlüssel dürfen nicht mit dem Präfix `aws:` beginnen.
- Schlüssel müssen in einem Tag-Satz eindeutig sein.
- Schlüssel müssen zwischen 1 und 128 Zeichen lang sein.
- Ein Wert muss zwischen 0 und 256 Zeichen haben.
- Werte brauchen pro Tag-Satz nicht eindeutig zu sein.
- Zulässige Zeichen für Schlüssel und Werte sind Buchstaben, Zahlen, Leerraum und eines der folgenden Symbole: _.:/= + - @.
- Bei Schlüsseln und Werten wird die Groß-/Kleinschreibung berücksichtigt.

Verwendungshinweise mit Tags versehen

Beachten Sie bei der Verwendung von Tags in Aurora DSQL Folgendes.

- Wenn Sie die DSQL-API-Operationen AWS CLI oder Aurora verwenden, stellen Sie sicher, dass Sie den Amazon-Ressourcennamen (ARN) angeben, mit dem die Aurora DSQL-Ressource arbeiten soll. Weitere Informationen finden Sie unter Amazon Resource Name (ARNs) -Format für Aurora DSQL-Ressourcen.
- Jede Ressource verfügt über genau einen Tag-Satz, d. h. eine Zusammenstellung von einem oder mehreren Tags, die der Ressource zugewiesen sind.
- Jede Ressource kann pro Tag-Satz bis zu 50 Tags enthalten.
- Wenn Sie eine Ressource löschen, werden alle Tags der Ressource ebenfalls gelöscht.
- Sie können Tags hinzufügen, wenn Sie eine Ressource erstellen. Sie können Tags mithilfe der folgenden API-Operationen anzeigen und ändern: `TagResource`, `UntagResource`, und `ListTagsForResource`.
- Sie können Tags mit IAM-Richtlinien verwenden. Sie können sie verwenden, um den Zugriff auf Aurora DSQL-Cluster zu verwalten und zu kontrollieren, welche Aktionen auf diese Ressourcen angewendet werden können. Weitere Informationen finden Sie unter Steuern des Zugriffs auf AWS Ressourcen mithilfe von Tags.
- Sie können Tags überall für verschiedene andere Aktivitäten verwenden AWS. Weitere Informationen finden Sie unter Allgemeine Tagging-Strategien.

Überlegungen zur Arbeit mit Amazon Aurora DSQL

Beachten Sie die folgenden Verhaltensweisen, wenn Sie mit Amazon Aurora DSQL arbeiten. Weitere Hinweise zur PostgreSQL-Kompatibilität und -Unterstützung finden Sie unter. Kompatibilität der SQL-Funktionen in Aurora DSQL Informationen zu Kontingenten und Beschränkungen finden Sie unter. Cluster-Kontingente und Datenbank-Limits in Amazon Aurora DSQL

- Aurora DSQL schließt `COUNT(*)` Operationen vor dem Transaktions-Timeout für große Tabellen nicht ab. Informationen zum Abrufen der Tabellenzeilenanzahl aus dem Systemkatalog finden Sie unter Verwenden von Systemtabellen und Befehlen in Aurora DSQL.
- Das Aufrufen von Treibern `PG_PREPARED_STATEMENTS` kann zu einer inkonsistenten Ansicht der zwischengespeicherten vorbereiteten Anweisungen für den Cluster führen. Möglicherweise sehen Sie mehr als die erwartete Anzahl vorbereiteter Anweisungen pro Verbindung für denselben Cluster und dieselbe IAM-Rolle. Aurora DSQL speichert keine von Ihnen vorbereiteten Anweisungsnamen.
- In seltenen Szenarien, in denen mehrere Regionen betroffen sind, kann es länger als erwartet dauern, bis die Transaktions-Commit-Verfügbarkeit wieder aufgenommen wird. Im Allgemeinen können automatisierte Cluster-Wiederherstellungsvorgänge zu vorübergehenden Parallelitätskontrollen oder Verbindungsfehlern führen. In den meisten Fällen werden Sie die Auswirkungen nur für einen bestimmten Prozentsatz Ihrer Arbeitslast sehen. Wenn Sie diese Übertragungsfehler sehen, wiederholen Sie Ihre Transaktion oder stellen Sie erneut eine Verbindung zu Ihrem Kunden her.
- Einige SQL-Clients, wie Datagrip, rufen umfangreiche Systemmetadaten auf, um Schemainformationen aufzufüllen. Aurora DSQL unterstützt nicht all diese Informationen und gibt Fehler zurück. Dieses Problem hat keinen Einfluss auf die SQL-Abfragefunktionalität, kann sich jedoch auf die Schemaanzeige auswirken.
- Die Administratorrolle verfügt über eine Reihe von Berechtigungen für Datenbankverwaltungsaufgaben. Standardmäßig gelten diese Berechtigungen nicht für Objekte, die andere Benutzer erstellen. Die Administratorrolle kann anderen Benutzern keine Berechtigungen für diese von Benutzern erstellten Objekte gewähren oder entziehen. Der Admin-Benutzer kann sich selbst jede andere Rolle zuweisen, um die erforderlichen Berechtigungen für diese Objekte zu erhalten.

Cluster-Kontingente und Datenbank-Limits in Amazon Aurora DSQL

In den folgenden Abschnitten werden die Cluster-Kontingente und Datenbanklimits für Aurora DSQL beschrieben.

Cluster-Kontingente

Ihr AWS-Konto hat die folgenden Cluster-Kontingente in Aurora DSQL. Um eine Erhöhung der Servicekontingente für Cluster mit einer oder mehreren Regionen innerhalb eines bestimmten Bereichs zu beantragen AWS-Region, verwenden Sie die Konsolenseite für Servicekontingente. Für weitere Kontingenterhöhungen wenden Sie sich bitte an. AWS -Support

Beschreibung	Standardlimit	Konfiguri erbar?	Aurora DSQL-Fehlercode
Maximale Anzahl von Clustern mit einer Region pro AWS-Konto	20 Cluster	Ja	API-Fehlercode `ServiceQuotaExceededExceptio`
Maximale Anzahl von Clustern mit mehreren Regionen pro AWS-Konto	5 Cluster	Ja	API-Fehlercode `ServiceQuotaExceededExceptio`
Maximaler Speicherplatz pro Cluster	10 TiB Standardl imit, bis zu 128 TiB mit genehmigt er Grenzwert erhöhung	Ja	`DISK_FULL(53100)`

Beschreibung	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode
Maximale Anzahl an Verbindungen pro Cluster	10.000 Verbindungen	Ja	`TOO_MANY_CONNECTIONS(53300)`
Maximale Verbindungsrate pro Cluster	100 Verbindungen pro Sekunde	Nein	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Maximale Verbindungs-Burst-Kapazität pro Cluster	1.000 Verbindungen	Nein	Kein Fehlercode
Maximale Anzahl gleichzeitiger Wiederherstellungsaufträge	4	Nein	Kein Fehlercode
Wiederauffüllrate der Verbindung	100 Verbindungen pro Sekunde	Nein	Kein Fehlercode

Datenbanklimits in Aurora DSQL

In der folgenden Tabelle werden die Datenbanklimits in Aurora DSQL beschrieben.

Beschreibung	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode	Fehlermeldung
Maximale kombinierte Größe der in einem Primärschlüssel verwendeten Spalten	1 KiB	Nein	54000	ERROR: key size too large
Maximale kombinierte Größe der Spalten in einem Sekundärindex	1 KiB	Nein	54000	ERROR: key size too large
Maximale Größe einer Zeile in einer Tabelle	2 MiB	Nein	54000	ERROR: maximum row size excee
Maximale Größe einer Spalte, die nicht Teil eines Indexes ist	1 MiB	Nein	54000	ERROR: maximum column size ex
Maximale Anzahl von Spalten in einem Primärschlüssel oder einem Sekundärindex	8	Nein	54011	ERROR: more than 8 column key are not supported
Maximale Anzahl von	255	Nein	54011	ERROR: tables can have at mos

Beschreibung	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode	Fehlermeldung
Spalten in einer Tabelle				
Maximale Anzahl von Indizes in einer Tabelle	24	Nein	`54000`	`ERROR: more than 24 indexes p allowed`
Maximale Größe aller Daten, die in einer Schreibtransaktion geändert wurden	10 MiB	Nein	`54000`	`ERROR: transaction size limit DETAIL: Current transaction s 10mb`
Maximale Anzahl von Tabellen- und Indexzeilen, die in einem Transaktionsblock mutiert werden können	3.000 Zeilen pro Transaktion. Siehe Überlegungen zu Aurora DSQL zur PostgreSQL-Kompatibilität.	Nein	`54000`	`ERROR: transaction row limit`
Maximale Basisspeichermenge, die ein Abfragevorgang verwenden kann	128 MiB pro Transaktion	Nein	`53200`	`ERROR: query requires too muc out of memory.`

Beschreibung	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode	Fehlermeldung
Maximale Anzahl von Schemas, die in einer Datenbank definiert sind	10	Nein	54000	ERROR: more than 10 schemas n
Maximale Anzahl von Tabellen in einer Datenbank	1.000 Tabellen	Nein	54000	ERROR: creating more than 100 allowed
Maximale Anzahl von Datenbanken in einem Cluster	1	Nein	Kein Fehlercode	ERROR: unsupported statement
Maximale Transaktionszeit	5 Minuten	Nein	54000	ERROR: transaction age limit exceeded
Maximale Verbindungsdauer	60 Minuten	Nein	Kein Fehlercode	Keine Fehlermeldung
Maximale Anzahl von Ansichten in einer Datenbank	5,000	Nein	54000	ERROR: creating more than 500 allowed
Maximale Größe der Ansichtsdefinition	2 MiB	Nein	54000	ERROR: view definition too la

Informationen zu spezifischen Datentypbeschränkungen für Aurora DSQL finden Sie unterUnterstützte Datentypen in Aurora DSQL.

Aurora DSQL API-Referenz

Neben dem AWS Management Console und dem AWS Command Line Interface (AWS CLI) bietet Aurora DSQL auch eine API-Schnittstelle. Sie können die API-Operationen verwenden, um Ihre Ressourcen in Aurora DSQL zu verwalten.

Eine alphabetische Liste der API-Operationen finden Sie unter Aktionen.

Eine alphabetische Liste der Datentypen finden Sie unter Datentypen.

Eine Liste der häufigen Abfrageparameter finden Sie unter Häufige Parameter.

Beschreibungen der Fehlercodes finden Sie unter Häufige Fehler.

Weitere Informationen zu finden Sie in der AWS CLI AWS Command Line Interface Referenz für Aurora DSQL.

Behebung von Problemen in Aurora DSQL

> **Note**
>
> Die folgenden Themen enthalten Hinweise zur Fehlerbehebung bei Fehlern und Problemen, die bei der Verwendung von Aurora DSQL auftreten können. Wenn Sie ein Problem finden, das hier nicht aufgeführt ist, wenden Sie sich an den Support AWS

Themen

Behebung von Verbindungsfehlern

Fehler: unbekannter SSL-Fehlercode: 6

Ursache: Möglicherweise verwenden Sie eine PSQL-Version vor Version 14, die Server Name Indication (SNI) nicht unterstützt. Das SNI ist erforderlich, wenn Sie eine Verbindung zu Aurora DSQL herstellen.

Sie können Ihre Client-Version mit überprüfen. `psql --version`

Fehler: NetworkUnreachable

Ein `NetworkUnreachable` Fehler bei Verbindungsversuchen könnte darauf hindeuten, dass Ihr Client keine IPv6 Verbindungen unterstützt, und nicht auf ein aktuelles Netzwerkproblem hindeuten. Dieser Fehler tritt aufgrund der Art und Weise, wie PostgreSQL-Clients IPv4 Dual-Stack-Verbindungen handhaben, häufig bei Instanzen auf, die nur die Option „-only“ haben. Wenn ein Server den Dual-Stack-Modus unterstützt, lösen diese Clients Hostnamen zunächst sowohl in Adressen als auch in Adressen auf. IPv4 IPv6 Sie versuchen zuerst, eine IPv4 Verbindung herzustellen, und versuchen es dann, IPv6 falls die erste Verbindung fehlschlägt. Wenn Ihr System

dies nicht unterstützt IPv6, wird anstelle der eindeutigen Meldung „IPv6 Nicht unterstützt“ ein allgemeiner `NetworkUnreachable` Fehler angezeigt.

Behebung von Authentifizierungsfehlern

Die IAM-Authentifizierung ist für den Benutzer „...“ fehlgeschlagen

Wenn Sie ein Aurora DSQL IAM-Authentifizierungstoken generieren, können Sie eine maximale Dauer von 1 Woche festlegen. Nach einer Woche können Sie sich nicht mit diesem Token authentifizieren.

Darüber hinaus lehnt Aurora DSQL Ihre Verbindungsanfrage ab, wenn Ihre angenommene Rolle abgelaufen ist. Wenn Sie beispielsweise versuchen, eine Verbindung mit einer temporären IAM-Rolle herzustellen, obwohl Ihr Authentifizierungstoken noch nicht abgelaufen ist, lehnt Aurora DSQL die Verbindungsanfrage ab.

Weitere Informationen darüber, wie IAM mit Aurora DSQL funktioniert, finden Sie unter Grundlegendes zur Authentifizierung und Autorisierung für Aurora DSQL und AWS Identity and Access Management in Aurora DSQL.

Beim Aufrufen des GetObject Vorgangs ist ein Fehler aufgetreten (InvalidAccessKeyId): Die von Ihnen angegebene AWS Zugriffsschlüssel-ID ist in unseren Aufzeichnungen nicht vorhanden

IAM hat Ihre Anfrage abgelehnt. Weitere Informationen finden Sie unter Warum Anfragen signiert werden.

Die IAM-Rolle ist nicht vorhanden <role>

Aurora DSQL konnte Ihre IAM-Rolle nicht finden. Weitere Informationen finden Sie unter IAM-Rollen.

Die IAM-Rolle muss wie ein IAM-ARN aussehen

Weitere Informationen finden Sie unter IAM-Identifikatoren — ARNs IAM.

Behebung von Autorisierungsfehlern

Rolle wird nicht unterstützt <role>

Aurora DSQL unterstützt den `GRANT` Vorgang nicht. Siehe Unterstützte Teilmengen von PostgreSQL-Befehlen in Aurora DSQL.

Eine Vertrauensstellung mit der Rolle kann nicht hergestellt werden <role>

Aurora DSQL unterstützt den `GRANT` Vorgang nicht. Siehe Unterstützte Teilmengen von PostgreSQL-Befehlen in Aurora DSQL.

Die Rolle ist nicht vorhanden <role>

Aurora DSQL konnte den angegebenen Datenbankbenutzer nicht finden. Weitere Informationen finden Sie unter Autorisieren benutzerdefinierter Datenbankrollen für die Verbindung zu einem Cluster.

FEHLER: Die Erlaubnis, der Rolle IAM eine Vertrauensstellung zu gewähren, wurde verweigert <role>

Um Zugriff auf eine Datenbankrolle zu gewähren, müssen Sie mit der Administratorrolle mit Ihrem Cluster verbunden sein. Weitere Informationen finden Sie unter Autorisieren von Datenbankrollen zur Verwendung von SQL in einer Datenbank.

FEHLER: Die Rolle muss das LOGIN-Attribut haben <role>

Alle Datenbankrollen, die Sie erstellen, müssen über die `LOGIN` entsprechende Berechtigung verfügen.

Um diesen Fehler zu beheben, stellen Sie sicher, dass Sie die PostgreSQL-Rolle mit der `LOGIN` entsprechenden Berechtigung erstellt haben. Weitere Informationen finden Sie unter CREATE ROLE und ALTER ROLE in der PostgreSQL-Dokumentation.

FEHLER: Die Rolle kann nicht gelöscht werden, da einige Objekte davon abhängen <role>

Aurora DSQL gibt einen Fehler zurück, wenn Sie eine Datenbankrolle mit einer IAM-Beziehung löschen, bis Sie die Beziehung mithilfe von widerrufen. `AWS IAM REVOKE` Weitere Informationen finden Sie unter Autorisierung widerrufen.

Behebung von SQL-Fehlern

Fehler: Nicht unterstützt

Aurora DSQL unterstützt nicht alle PostgreSQL-basierten Dialekte. Informationen darüber, was unterstützt wird, finden Sie unter Unterstützte PostgreSQL-Funktionen in Aurora DSQL.

Fehler: SELECT FOR UPDATE in einer schreibgeschützten Transaktion ist ein No-Op

Sie versuchen einen Vorgang, der in einer schreibgeschützten Transaktion nicht zulässig ist. Weitere Informationen finden Sie unter Grundlegendes zur Parallelitätssteuerung in Aurora DSQL.

Fehler: Verwenden Sie stattdessen **`CREATE INDEX ASYNC`**

Um einen Index für eine Tabelle mit vorhandenen Zeilen zu erstellen, müssen Sie den `CREATE INDEX ASYNC` Befehl verwenden. Weitere Informationen finden Sie unter Asynchrone Erstellung von Indizes in Aurora DSQL.

Behebung von OCC-Fehlern

OC000 „FEHLER: Die Mutation steht in Konflikt mit einer anderen Transaktion, versuchen Sie es bei Bedarf erneut“

OC001 „FEHLER: Das Schema wurde durch eine andere Transaktion aktualisiert, versuchen Sie es bei Bedarf erneut“

Ihre PostgreSQL-Sitzung hatte eine zwischengespeicherte Kopie des Schemakatalogs. Diese zwischengespeicherte Kopie war zum Zeitpunkt des Ladens gültig. Nennen wir die Zeit T1 und die Version V1.

Eine weitere Transaktion aktualisiert den Katalog zum Zeitpunkt T2. Nennen wir das V2.

Wenn die ursprüngliche Sitzung zum Zeitpunkt T2 versucht, aus dem Speicher zu lesen, verwendet sie immer noch die Katalogversion V1. Die Speicherschicht von Aurora DSQL lehnt die Anfrage ab, da die neueste Katalogversion bei T2 V2 ist.

Wenn Sie zum Zeitpunkt T3 der ursprünglichen Sitzung erneut versuchen, aktualisiert Aurora DSQL den Katalog-Cache. Die Transaktion bei T3 verwendet Katalog V2. Aurora DSQL wird die Transaktion abschließen, solange seit dem Zeitpunkt T2 keine weiteren Katalogänderungen vorgenommen wurden.

Fehlerbehebung bei Verbindungen SSL/TLS

SSL-Fehler: Die Überprüfung des Zertifikats ist fehlgeschlagen

Dieser Fehler weist darauf hin, dass der Client das Serverzertifikat nicht verifizieren kann. Stellen Sie sicher, dass:

1. Das Amazon Root CA 1-Zertifikat ist ordnungsgemäß installiert. Anweisungen Konfiguration von SSL/TLS Zertifikaten für Aurora DSQL-Verbindungen zur Validierung und Installation dieses Zertifikats finden Sie unter.
2. Die `PGSSLROOTCERT` Umgebungsvariable verweist auf die richtige Zertifikatsdatei.
3. Die Zertifikatsdatei hat die richtigen Berechtigungen.

Unbekannter SSL-Fehlercode: 6

Dieser Fehler tritt bei PostgreSQL-Clients unter Version 14 auf. Aktualisieren Sie Ihren PostgreSQL-Client auf Version 17, um dieses Problem zu beheben.

SSL-Fehler: nicht registriertes Schema (Windows)

Dies ist ein bekanntes Problem mit dem Windows-Psql-Client bei der Verwendung von Systemzertifikaten. Verwenden Sie die in der Verbindung von Windows aus herstellen Anleitung beschriebene Methode zur heruntergeladenen Zertifikatsdatei.

Dokumentenverlauf für das Amazon Aurora DSQL-Benutzerhandbuch

In der folgenden Tabelle werden die Dokumentationsversionen für Aurora DSQL beschrieben.

Änderung	Beschreibung	Datum
Allgemeine Verfügbarkeit (GA) von Amazon Aurora DSQL	Amazon Aurora DSQL ist jetzt allgemein verfügbar und bietet zusätzliche Unterstützung für CloudWatch Überwachung, erweiterte Datenschutzfunktionen und AWS Backup Integration. Weitere Informationen finden Sie unter Überwachung von Aurora DSQL mit CloudWatch, Backup und Wiederherstellung für Amazon Aurora DSQL und Datenverschlüsselung für Amazon Aurora DSQL.	27. Mai 2025
AmazonAuroraDSQLFullZugriffsupdate	Fügt die Möglichkeit hinzu, Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster durchzuführen, einschließlich Starten, Stoppen und Überwachen von Jobs. Es bietet auch die Möglichkeit, vom Kunden verwaltete KMS-Schlüssel für die Clusterverschlüsselung zu verwenden. Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugreifen und Verwenden	21. Mai 2025

	von serviceverknüpften Rollen in Aurora DSQL.	
AmazonAuroraDSQLConsoleFullAccess update	Fügt die Möglichkeit hinzu, Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster über die AWS Console Home durchzuführen. Dies beinhaltet das Starten, Stoppen und Überwachen von Jobs. Es unterstützt auch die Verwendung von vom Kunden verwalteten KMS-Schlüsseln für die Clusterverschlüsselung und den Start AWS CloudShell. Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	21. Mai 2025

AmazonAuroraDSQLReadOnlyAccess aktualisieren	Beinhaltet die Möglichkeit, den richtigen VPC-Endpunkt-Servicenamen zu ermitteln, wenn über AWS PrivateLink Aurora DSQL eine Verbindung zu Ihren Aurora DSQL-Clustern hergestellt wird. Dadurch werden eindeutige Endpunkte pro Zelle erstellt, sodass diese API sicherstellt, dass Sie den richtigen Endpunkt für Ihren Cluster identifizieren und Verbindungsfehler vermeiden können. Weitere Informationen finden Sie unter AmazonAuroraDSQLReadOnlyAccessund Verwenden von servicegebundenen Rollen in Aurora DSQL.	13. Mai 2025

AmazonAuroraDSQLFullZugriffsupdate	Die Richtlinie fügt vier neue Berechtigungen zum Erstellen und Verwalten von Datenbankclustern für mehrere AWS-Regionen:`PutMultiRegionProperties` , `PutWitnessRegion` `AddPeerCluster` , und hinzu`RemovePeerCluster` . Zu diesen Berechtigungen gehören Steuerungen auf Ressourcenebene und Bedingungsschlüssel, sodass Sie steuern können, welche Cluster-Benutzer Sie ändern können. Die Richtlinie fügt auch die `GetVpcEndpointServiceName` Erlaubnis hinzu, Ihnen dabei zu helfen, eine Verbindung zu Ihren Aurora DSQL-Clustern herzustellen. AWS PrivateLink Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025

AmazonAuroraDSQLConsoleFullAccess update	Fügt Aurora DSQL neue Berechtigungen hinzu, um Clustermanagement in mehreren Regionen und VPC-Endpunktverbindungen zu unterstützen. Zu den neuen Berechtigungen gehören:. `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025
AuroraDsqlServiceLinkedRolePolicy update	Fügt die Möglichkeit hinzu, Metriken in der Richtlinie `AWS/AuroraDSQL` und in `AWS/Usage` CloudWatch Namespaces zu veröffentlichen. Dadurch kann der zugehörige Dienst oder die zugehörige Rolle umfassendere Nutzungs- und Leistungsdaten an Ihre Umgebung senden. CloudWatch Weitere Informationen finden Sie unter AuroraDsqlServiceLinkedRolePolicyund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	8. Mai 2025

AWS PrivateLink für Amazon Aurora DSQL	Aurora DSQL unterstützt AWS PrivateLink jetzt. Mit AWS PrivateLink können Sie die private Netzwerkkonnektivität zwischen virtuellen privaten Clouds (VPCs), Aurora DSQL und Ihren lokalen Rechenzentren mithilfe von Amazon VPC-Endpunkten und privaten IP-Adressen vereinfachen. Weitere Informationen finden Sie unter Verwaltung und Verbindung zu Amazon Aurora DSQL-Clustern mithilfe von AWS PrivateLink.	8. Mai 2025
Erstversion	Erste Version des Amazon Aurora DSQL-Benutzerhandbuchs.	3. Dezember 2024

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.