aws

Benutzer-Leitfaden

Amazon Aurora DSQL

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

Amazon Aurora DSQL: Benutzer-Leitfaden

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist Amazon Aurora DSQL? ... 1
Wann sollte dies verwendet werden? ... 1
Schlüssel-Features ... 1
AWS-Region Verfügbarkeit ... 3
Cluster mit mehreren Regionen ... 5
Preisgestaltung ... 6
Als nächstes ... 6
Erste Schritte ... 7
Voraussetzungen ... 7
Erstellen Sie ein Einzelregions-Cluster ... 7
Verbinden mit einem Cluster ... 8
SQL-Befehle ausführen ... 9
Erstellen eines Clusters mit mehreren Regionen ... 10
Fehlerbehebung ... 13
Authentifizierung und Autorisierung ... 14
Verwalten Ihrer Cluster ... 14
Verbindungsaufbau mit Ihrem Cluster ... 14
PostgreSQL- und IAM-Rollen ... 15
IAM-Richtlinienaktionen in Aurora DSQL verwenden ... 16
Verwenden von IAM-Richtlinienaktionen zur Verbindung mit Clustern ... 17
Verwenden von IAM-Richtlinienaktionen zur Clusterverwaltung ... 17
Widerrufen der Autorisierung mithilfe von IAM und PostgreSQL ... 18
Erstellen eines Authentifizierungstokens ... 19
Konsole ... 20
AWS CloudShell ... 20
AWS CLI ... 22
Aurora SQL SDKs ... 23
Datenbankrollen und IAM-Authentifizierung ... 32
IAM-Rollen ... 32
IAM-Benutzer ... 33
Verbinden ... 33
Query ... 33
Zuordnungen anzeigen ... 34
Widerrufen ... 35

Aurora DSQL und PostgreSQL 36
Kompatibilitätsmerkmale 36
Vorteile einer verteilten Architektur 37
SQL-Kompatibilität 38
Unterstützte Datentypen 38
Unterstützte SQL-Features 44
Unterstützte Teilmengen von SQL-Befehlen 48
Migrationshandbuch 70
Steuerelemente für die Parallelität 78
Transaktionskonflikte 79
Richtlinien für die Optimierung der Transaktionsleistung 79
DDL und verteilte Transaktionen 79
Primärschlüssel 81
Datenstruktur und Speicherung 81
Richtlinien für die Auswahl eines Primärschlüssels 81
Sequenzen und Identitätsspalten 82
Sequenz-Bearbeitungsfunktionen 83
Identitätsspalten 86
Arbeiten mit Sequenzen und Identitätsspalten 87
Asynchrone Indizes 89
Syntax 90
Parameters 90
Nutzungshinweise 91
Erstellen eines -Index 92
Abfragen eines Index 93
Fehler beim Erstellen eines eindeutigen Indexes 94
Eindeutigkeitsverstöße 94
Systemtabellen und Befehle 96
Systemtabellen 97
Nützliche Systemabfragen 107
Der ANALYZE-Befehl. 108
PLÄNE ERKLÄREN 109
PostgreSQL EXPLAIN-Pläne 109
Die wichtigsten Elemente 111
Filtern 111
EXPLAIN-Pläne lesen 112

DPUs in EXPLAIN ANALYZE 116
Verwalten von Aurora-DB-Clustern 121
Cluster mit einer einzelnen Region 121
AWS-SDKs verwenden 121
Verwenden von AWS-CLI 160
Cluster mit mehreren Regionen 164
AWS-SDKs verwenden 164
AWS CLI verwenden 218
CloudFormation 224
Anfängliche Konfiguration 225
Nach Clustern suchen 225
Konfiguration aktualisieren 226
Aurora DSQL-Cluster-Lebenszyklus 226
Clusterstatus 226
Anzeigen des Clusterstatus 229
Programmieren mit Aurora DSQL 231
Konnektoren 231
JDBC-Konnektor 232
Python-Konnektor 237
Go-Konnektor 249
Node.js -Anschlüsse 256
Ruby-Konnektor 265
.NET-Konnektor 271
Zugreifen auf Aurora SQL 277
SQL-Clients 278
DBeaver 278
JetBrains DataGrip 282
Psql 283
VSCode 285
Fehlerbehebung 286
Datenbankverbindungstools 287
Aurora DSQL-Adapter 287
Beispiele für Datenbanktreiber 288
ORM- und Framework-Beispiele 289
Laden von Daten 291
Wählen Sie einen Ladeansatz 291

Aurora SQL-Lader ... 292
Migrationspfade ... 297
PostgreSQL\ copy verwenden ... 299
Weitere Ressourcen ... 300
Generative KI ... 300
AWS Labs Aurora DSQL-MCP-Server ... 301
Aurora DSQL Steering: Fähigkeiten und Fähigkeiten ... 309
Abfrage-Editor ... 315
Voraussetzungen ... 315
Arbeiten mit dem Abfrage-Editor ... 316
Abfrage-Editoren: Verwendung JupyterLab mit Aurora DSQL ... 318
Erste Schritte ... 318
Beispiel für ein Notizbuch ... 321
Weitere Informationen ... 321
Backup und Wiederherstellung ... 322
Erste Schritte mit AWS Backup ... 322
Wiederherstellen Ihrer Backups ... 322
Wiederherstellen von Clustern für eine einzelne Region ... 323
Wiederherstellen von Clustern mit mehreren Regionen ... 323
Überwachung und Compliance ... 323
Weitere Ressourcen ... 324
Überwachung und Protokollierung ... 325
Überwachung mit CloudWatch ... 325
Beobachtbarkeit ... 325
Verwendung ... 327
Protokollierung mit CloudTrail ... 328
Verwaltungsereignisse ... 329
Datenereignisse ... 330
Sicherheit ... 332
AWS verwaltete Richtlinien ... 333
AmazonAuroraDSQLFullZugriff ... 333
AmazonAuroraDSQLReadOnlyAccess ... 335
AmazonAuroraDSQLConsoleFullAccess ... 335
Aurora DSQLService RolePolicy ... 337
Richtlinienaktualisierungen ... 337
Datenschutz ... 345

Datenverschlüsselung ... 346
Datenschutz in Witness-Regionen ... 348
SSL-/TLS-Zertifikate ... 348
Datenverschlüsselung ... 346
KMS-Schlüsseltypen ... 355
Verschlüsselung im Ruhezustand ... 356
Verwenden von KMS und Datenschlüsseln ... 358
Autorisieren Ihres KMS-Schlüssels ... 360
Verschlüsselungskontext ... 362
Überwachung AWS KMS ... 363
Einen verschlüsselten Cluster erstellen ... 365
Einen Schlüssel entfernen oder aktualisieren ... 368
Überlegungen ... 370
Identity and Access Management ... 371
Zielgruppe ... 371
Authentifizierung mit Identitäten ... 372
Verwalten des Zugriffs mit Richtlinien ... 373
Funktionsweise von Amazon DSQL mit IAM ... 375
Beispiele für identitätsbasierte Richtlinien ... 381
Fehlerbehebung ... 387
Ressourcenbasierte Richtlinien ... 389
Wann sollte dies verwendet werden? ... 390
Mit Richtlinien erstellen ... 391
Richtlinien hinzufügen und bearbeiten ... 394
Richtlinie anzeigen ... 397
Richtlinie entfernen ... 398
Beispiele für Richtlinien ... 400
Blockieren des öffentlichen Zugriffs ... 404
API-Operationen ... 408
Verwenden einer serviceverknüpften Rolle ... 410
Serviceverknüpfte Rollenberechtigungen für Aurora DSQL ... 411
Erstellen einer serviceverknüpften Rolle ... 412
Bearbeiten einer serviceverknüpften Rolle ... 412
Löschen Sie eine serviceverknüpfte Rolle ... 412
Unterstützte Regionen für serviceverknüpfte Aurora DSQL-Rollen ... 412
Verwenden von IAM-Bedingungsschlüsseln ... 412

Erstellen Sie einen Cluster in einer bestimmten Region ... 413
Einen multiregionalen Cluster in bestimmten Regionen erstellen ... 413
Einen Cluster mit mehreren Regionen mit einer bestimmten Witness-Region erstellen ... 414
Vorfallreaktion ... 415
Compliance-Validierung ... 416
Ausfallsicherheit ... 416
Backup und Wiederherstellung ... 417
Replikation ... 417
Hohe Verfügbarkeit ... 418
Testen mit Fehlerinjektionen ... 418
Infrastruktursicherheit ... 419
Verwaltung von Clustern mit AWS PrivateLink ... 420
Konfigurations- und Schwachstellenanalyse ... 431
Serviceübergreifende Confused-Deputy-Prävention ... 432
Bewährte Methoden für die Gewährleistung der Sicherheit ... 433
Bewährte Methoden für aufdeckende Sicherheitsmaßnahmen ... 434
Bewährte Methoden für vorbeugende Sicherheitsmaßnahmen ... 435
Taggen von -Ressourcen ... 437
Namens-Tag ... 437
Anforderungen zum Markieren ... 437
Hinweise zur Verwendung von Tags ... 438
Überlegungen ... 439
Kontingente und -Einschränkungen ... 441
Cluster-Kontingente ... 441
Datenbanklimits ... 442
API-Referenz ... 447
Fehlerbehebung ... 280
Verbindungsfehler ... 448
Authentifizierungsfehler ... 449
Autorisierungsfehler ... 450
SQL-Fehler ... 451
OCC-Fehler ... 451
SSL-/TLS-Verbindungen ... 452
Geben von Feedback ... 453
Feedback-Kanäle ... 453
Effektive Funktionsanfragen ... 453

Dokumentverlauf 454
..... cdlxxiv

Was ist Amazon Aurora DSQL?

Amazon Aurora DSQL ist eine verteilte, relationale Serverless-Datenbank, die für transaktionale Workloads optimiert ist. Aurora DSQL bietet praktisch unbegrenzte Skalierbarkeit und erfordert keine manuelle Infrastrukturverwaltung. Die hochverfügbare 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 serverless ist, eignet sich Aurora DSQL ideal für Anwendungsmuster von Microservice-, ereignisgesteuerten und Serverless-Architekturen. Aurora DSQL ist PostgreSQL-kompatibel, sodass Sie vertraute Treiber, objektrelationale Mappings (), Frameworks und SQL-Funktionen verwenden können. ORMs

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

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

Die wichtigsten Features von Aurora DSQL

Die folgenden Hauptfeatures helfen Ihnen bei der Erstellung einer verteilten Serverless-Datenbank zur Unterstützung Ihrer Hochverfügbarkeitsanwendungen:

Verteilte Architektur

Aurora DSQL besteht aus den folgenden Multi-Tenant-Komponenten:

- Relay und Konnektivität

- Datenverarbeitung (Compute) und Datenbanken
- Transaktionsprotokoll, Parallelitätskontrolle und Isolierung
- Speicher

Eine Steuerebene koordiniert die oben angeführten Komponenten. Jede Komponente bietet Redundanz über drei Availability Zones (AZs) mit automatischer Cluster-Skalierung und Selbstheilung bei Komponentenausfällen. Weitere Informationen dazu, wie diese Architektur Hochverfügbarkeit unterstützt, finden Sie unter Ausfallsicherheit in Amazon Aurora DSQL.

Cluster mit einer Region und mehreren Regionen

Aurora DSQL-Cluster bieten die folgenden Vorteile:

- Synchrone Datenreplikation
- Konsistente Lesevorgänge
- Automatische Fehlerwiederherstellung
- 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 atomare, konsistente, isolierte und haltbare (ACID)-Transaktionen mit starker Konsistenz, Snapshot-Isolation, Atomarität sowie AZ-übergreifender und regionaler Beständigkeit.

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 bieten eine einzige logische Datenbank an. Sie sind für gleichzeitige Lese- und Schreibvorgänge verfügbar und bieten eine hohe Datenkonsistenz. Sie können Anwendungen entwickeln, die gleichzeitig in mehreren Regionen ausgeführt werden – für hohe Leistung und Ausfallsicherheit – und dabei sicherstellen, dass Leser stets dieselben Daten sehen.

Kompatibilität mit PostgreSQL

Die verteilte Datenbankebene (Compute) in Aurora DSQL basiert auf einer aktuellen Hauptversion von PostgreSQL. Sie können sich mit gängigen PostgreSQL-Treibern und Tools, z. B. `psql`, mit Aurora DSQL verbinden. Aurora DSQL ist derzeit mit PostgreSQL Version 16 kompatibel und unterstützt eine Vielzahl von PostgreSQL-Funktionen, Ausdrücken und Datentypen. Weitere Informationen zu den unterstützten SQL-Features finden Sie unter Kompatibilität der SQL-Features in Aurora DSQL.

Regionale Verfügbarkeit von 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 Regionenverfügbarkeit bestimmt, wo Sie Aurora DSQL-Datenbank-Cluster erstellen und verwalten können. Datenbankadministratoren und Anwendungsarchitekten, die hochverfügbare, global verteilte Datenbanksysteme entwerfen, müssen die Regionenunterstützung für ihre Workloads genau kennen. Zu den häufigsten Anwendungsfällen gehören die Einrichtung einer regionsübergreifenden Notfallwiederherstellung, die Bereitstellung für Benutzer aus geografisch näher gelegenen Datenbankinstances 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.

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
USA Ost (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS	
		dsql-fips.us-east-2.api.aws	HTTPS	
USA Ost (Nord-Virginia)	us-east-1	dsql.us-east-1.api.aws	HTTPS	
		dsql-fips.us-east-1.api.aws	HTTPS	
USA West (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS	
		dsql-fips.us-west-2.api.aws	HTTPS	
Asien-Pazifik (Melbourne)	ap-southeast-4	dsql.ap-southeast-4.api.aws	HTTPS	

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS
Asien-Pazifik (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asien-Pazifik (Sydney)	ap-southeast-2	dsql.ap-southeast-2.api.aws	HTTPS
Asien-Pazifik (Tokio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Kanada (Zentral)	ca-central-1	dsql.ca-central-1.api.aws dsql-fips.ca-central-1.api.aws	HTTPS HTTPS
Kanada West (Calgary)	ca-west-1	dsql.ca-west-1.api.aws dsql-fips.ca-west-1.api.aws	HTTPS HTTPS
Europa (Frankfurt)	eu-central-1	dsql.eu-central-1.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

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
Europa (Paris)	eu-west-3	dsql.eu-west-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

- Europe (Frankfurt)
- Europa (Irland)
- Europa (London)
- Europa (Paris)

Wichtige Einschränkungen

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

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

Preisgestaltung für Aurora DSQL

Kosteninformationen finden Sie unter Aurora DSQL-Preise.

Als nächstes

Weitere Informationen zu den Kernkomponenten von Aurora DSQL und zum Einstieg in den Service finden Sie unter den folgenden Links:

- Erste Schritte mit Aurora DSQL
- Kompatibilität der SQL-Features in Aurora DSQL
- Zugreifen auf Aurora DSQL mit PostgreSQL-kompatiblen Clients
- Aurora DSQL und PostgreSQL

Erste Schritte mit Aurora DSQL

Amazon Aurora DSQL ist eine serverlose, vollständig verwaltete, 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 der AWS Konsole auf Cluster zu und interagieren optional mit Ihrer Datenbank über andere PostgreSQL-Clients. Am Ende haben Sie einen funktionierenden Aurora DSQL-Cluster eingerichtet, der für Test- oder Produktionsworkloads einsatzbereit ist.

Topics

- Voraussetzungen
- Schritt 1: Erstellen eines Einzelregions-Clusters mit Aurora DSQL
- Schritt 2: Stellen Sie eine Verbindung zu Ihrem Aurora DSQL-Cluster her
- Schritt 3: Führen Sie SQL-Beispielbefehle in Aurora DSQL aus
- Schritt 4 (optional): Erstellen Sie einen Cluster mit mehreren Regionen
- Fehlerbehebung

Voraussetzungen

Bevor Sie die Arbeit mit Aurora DSQL aufnehmen, prüfen Sie, ob die folgenden Voraussetzungen erfüllt sind:

- Ihre IAM-Identität muss berechtigt sein, sich an der Konsole anzumelden.
- Ihre IAM-Identität muss die folgenden Kriterien erfüllen:
  - Zugriff zur Ausführung beliebiger Aktionen auf einer beliebigen Ressource in Ihrem AWS-Konto
  - `AmazonAuroraDSQLConsoleFullAccess` AWS Eine verwaltete Richtlinie ist beigefügt.

Schritt 1: Erstellen eines Einzelregions-Clusters mit Aurora DSQL

Die Grundeinheit von Aurora DSQL ist der Cluster, in dem Ihre Daten gespeichert werden. In dieser Aufgabe erstellen Sie einen Cluster in einer einzelnen AWS-Region.

So erstellen Sie einen Cluster mit einer einzelnen Region in Aurora DSQL

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Klicken Sie auf Cluster erstellen und dann auf Einzelregion.
3. (Optional) Ändern Sie den Wert des Standard-Name-Tags.
4. (Optional) Fügen Sie zusätzliche Tags für diesen Cluster hinzu.
5. (Optional) Wählen Sie in den Clustereinstellungen eine der folgenden Optionen aus:
   - Wählen Sie Verschlüsselungseinstellungen anpassen (Erweitert) aus, um einen AWS KMS key auszuwählen oder zu erstellen. Wenn Sie einen vom Kunden verwalteten Schlüssel verwenden, stellen Sie sicher, dass die Schlüsselrichtlinie Aurora DSQL die erforderlichen Berechtigungen gewährt. Weitere Informationen finden Sie unter Schlüsselrichtlinie für einen kundenseitig verwalteten Schlüssel.
   - Wählen Sie Löschschutz aktivieren aus, um zu verhindern, dass eine Löschoperation Ihren Cluster entfernt. Der Löschschutz ist standardmäßig ausgewählt.
   - Wählen Sie Ressourcenbasierte Richtlinie (erweitert) aus, um Zugriffskontrollrichtlinien für diesen Cluster festzulegen.
6. Wählen Sie Cluster erstellen.
7. Die Konsole führt Sie zurück zur Seite „Cluster“. Es wird ein Benachrichtigungsbanner angezeigt, das darauf hinweist, dass der Cluster erstellt wird. Wählen Sie die Cluster-ID aus, um die Cluster-Detailansicht zu öffnen.

Schritt 2: Stellen Sie eine Verbindung zu Ihrem Aurora DSQL-Cluster her

Aurora SQL unterstützt mehrere Möglichkeiten, eine Verbindung zu Ihrem Cluster herzustellen, darunter den SQL Query Editor AWS CloudShell, den lokalen PSQL-Client und andere PostgreSQL-kompatible Tools. In diesem Schritt stellen Sie mit dem Aurora DSQL Query Editor eine Verbindung her, der eine schnelle Möglichkeit bietet, mit der Interaktion mit Ihrem neuen Cluster zu beginnen.

So stellen Sie mit dem Abfrage-Editor eine Verbindung her

1. Öffnen Sie in der Aurora DSQL Console (https://console.aws.amazon.com/dsql) die Cluster-Seite und vergewissern Sie sich, dass Ihre Cluster-Erstellung abgeschlossen ist und der Status Aktiv lautet.
2. Wählen Sie Ihren Cluster aus der Liste aus oder wählen Sie die Cluster-ID, um die Cluster-Detailseite zu öffnen.
3. Wählen Sie Mit dem Abfrage-Editor Connect.
4. Wählen Sie Connect as admin für den Cluster, der gerade erstellt wurde.
   - Optional können Sie eine Verbindung mit einer benutzerdefinierten Rolle herstellen, sieheVerwenden von Datenbankrollen und IAM-Authentifizierung.

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

Testen Sie Ihren Aurora DSQL-Cluster, indem Sie SQL-Anweisungen ausführen. Nachdem Sie den Cluster im Abfrage-Editor geöffnet haben, wählen Sie jede Beispielabfrage aus und führen Sie sie Schritt für Schritt aus.

Führen Sie SQL-Beispielbefehle in Aurora DSQL aus

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

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

2. Erstellen Sie eine hello_world-Tabelle, die eine automatisch generierte UUID als Primärschlüssel verwendet.

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

3. Fügen Sie eine Beispielzeile ein.

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

4. Lesen Sie die eingefügten Werte.

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

5. Optional bereinigen

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

Schritt 4 (optional): Erstellen Sie einen Cluster mit mehreren Regionen

Wenn Sie ein Cluster mit mehreren Regionen in Aurora DSQL erstellen, geben Sie die folgenden Regionen an:

Remote-Region

In dieser Region erstellen Sie einen zweiten Cluster, der mit Ihrem ursprünglichen Cluster verbunden wird. Sie erstellen einen zweiten Cluster in dieser Region und führen ein Peering mit Ihrem ursprünglichen Cluster aus. Aurora DSQL repliziert alle Schreibvorgänge vom ursprünglichen Cluster zum Remote-Cluster. Sie können auf jedem Cluster lesen und schreiben.

Witness-Region

Diese Region empfängt alle Daten, die in den Cluster mit mehreren Regionen geschrieben werden. Witness-Regionen hosten jedoch keine Client-Endpunkte und bieten keinen Zugriff auf Benutzerdaten. Es wird ein verschlüsseltes Transaktionsprotokoll mit begrenztem Zeitfenster gespeichert. Dieses Protokoll unterstützt die Wiederherstellung und stellt ein transaktionales Quorum bereit, falls eine Region ausfällt.

Gehen Sie wie folgt vor, um einen ersten Cluster zu erstellen, einen zweiten Cluster in einer anderen Region zu erstellen und dann die beiden Cluster miteinander zu verbinden, um einen Cluster mit mehreren Regionen zu erstellen. Des Weiteren werden die regionsübergreifende Schreibreplikation sowie konsistente Lesevorgänge von beiden regionalen Endpunkten demonstriert.

Erstellen eines Clusters mit mehreren Regionen

1. Melden Sie sich bei der Aurora DSQL-Konsole an.

2. Klicken Sie im Navigationsbereich auf Cluster.
3. Klicken Sie auf Cluster erstellen und wählen Sie dann Mehrere Region aus.
4. (Optional) Ändern Sie den Wert des Standard-Name-Tags.
5. (Optional) Fügen Sie zusätzliche Tags für diesen Cluster hinzu.
6. Wählen Sie unter 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 Witness-Regionen 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.

7. (Optional) Wählen Sie in den Clustereinstellungen eine der folgenden Optionen für Ihren ersten Cluster aus:

   - Wählen Sie Verschlüsselungseinstellungen anpassen (Erweitert) aus, um einen AWS KMS key auszuwählen oder zu erstellen. Wenn Sie einen vom Kunden verwalteten Schlüssel verwenden, stellen Sie sicher, dass die Schlüsselrichtlinie Aurora DSQL die erforderlichen Berechtigungen gewährt. Weitere Informationen finden Sie unter Schlüsselrichtlinie für einen kundenseitig verwalteten Schlüssel.
   - Wählen Sie Löschschutz aktivieren aus, um zu verhindern, dass eine Löschoperation Ihren Cluster entfernt. Der Löschschutz ist standardmäßig ausgewählt.
   - Wählen Sie Ressourcenbasierte Richtlinie (erweitert) aus, um Zugriffskontrollrichtlinien für diesen Cluster festzulegen.

8. Klicken Sie auf Cluster erstellen, um Ihren ersten Cluster zu erstellen. Wenn Sie im vorherigen Schritt keinen ARN eingegeben haben, zeigt die Konsole die Benachrichtigung Cluster-Setup steht aus an.
9. Klicken Sie in der Benachrichtigung Cluster-Setup steht aus auf Vollständiges Cluster-Setup für mehrere Regionen. Diese Aktion initiiert die Erstellung eines zweiten Clusters in einer anderen Region.
10. Wählen Sie für Ihren zweiten Cluster eine der folgenden Optionen aus:

- Cluster-ARN für Remote-Regionen hinzufügen – Wählen Sie diese Option, wenn bereits ein Cluster vorhanden ist und Sie möchten, dass dieser der zweite Cluster in Ihrem Cluster mit mehreren Regionen wird.
- 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.

11. 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.
12. (Optional) Auswählen der Clustereinstellungen für Ihren zweiten Clusters. Sie können beispielsweise einen AWS KMS key auswählen. Wenn Sie einen vom Kunden verwalteten Schlüssel verwenden, stellen Sie sicher, dass die Schlüsselrichtlinie Aurora DSQL die erforderlichen Berechtigungen gewährt. Weitere Informationen finden Sie unter Schlüsselrichtlinie für einen kundenseitig verwalteten Schlüssel.
13. Klicken Sie auf Cluster erstellen, um jetzt Ihren zweiten Cluster zu erstellen.
14. 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.
15. Klicken Sie zur Bestätigung auf Bestätigen. Mit diesem Schritt ist die Erstellung Ihres Clusters mit mehreren Regionen abgeschlossen.

Verbindung zu Ihrem zweiten Cluster herstellen

1. Öffnen Sie die Aurora DSQL-Konsole und wählen Sie die Region für Ihren zweiten Cluster aus.
2. Klicken Sie auf Cluster.
3. Wählen Sie die Zeile für den zweiten Cluster in Ihrem Cluster mit mehreren Regionen aus.
4. Wählen Sie Mit dem Abfrage-Editor Connect.
5. Klicken Sie auf Als Admin verbinden.
6. Erstellen Sie ein Beispielschema und eine Beispieltabelle und fügen Sie Daten ein, indem Sie die Schritte unter befolgenSchritt 3: Führen Sie SQL-Beispielbefehle in Aurora DSQL aus.

So fragen Sie Daten im zweiten Cluster aus der Region ab, in der Ihr erster Cluster gehostet wird

1. ählen Sie die Region für Ihren ersten Cluster in der Aurora DSQL-Konsole aus.
2. Klicken Sie auf Cluster.
3. Wählen Sie die Zeile für den zweiten Cluster in Ihrem Cluster mit mehreren Regionen aus.

4. Wählen Sie Mit dem Abfrage-Editor Connect.
5. Klicken Sie auf Als Admin verbinden.
6. Fragen Sie die Daten ab, die Sie in den zweiten Cluster eingefügt haben.

   Example

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

Fehlerbehebung

Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Aurora DSQL-Dokumentation.

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-Berechtigungen. Aurora DSQL verwendet diese Features, um eine umfassende Autorisierungs- und Zugriffsrichtlinie für Ihren Cluster, Ihre Datenbank und Ihre Daten bereitzustellen.

Verwalten Ihres Clusters mithilfe von IAM

Zur Verwaltung Ihres Clusters verwenden Sie IAM für die Authentifizierung und Autorisierung:

IAM-Authentifizierung

Zur Authentifizierung Ihrer IAM-Identität bei der Verwaltung von Aurora DSQL-Clustern müssen Sie IAM verwenden. Sie können sich über die AWS-Managementkonsole, über AWS CLI oder über AWS SDK authentifizieren.

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 `dsql:GetCluster` wie in der folgenden Beispiel-Richtlinienaktion dargestellt verfügt.

```
{
  "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 Clusterverwaltung.

Verbindungsaufbau mit Ihrem Cluster mithilfe von IAM

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 der Autorisierung, eine Verbindung zu Ihrem Cluster herzustellen. Weitere Informationen hierzu finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

IAM-Autorisierung

Gewähren Sie der IAM-Identität, mit der Sie die Verbindung zum Endpunkt Ihres Clusters herstellen, die Erlaubnis für folgende 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`, wenn Sie eine benutzerdefinierte Datenbankrolle verwenden. Diese Rolle erstellen und verwalten Sie mithilfe von SQL-Befehlen in Ihrer Datenbank. Mit der folgenden IAM-Richtlinienaktion kann eine benutzerdefinierte Datenbankrolle bis zu einer Stunde *`my-cluster`* lang eine Verbindung herstellen.

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

Nachdem Sie eine Verbindung hergestellt haben, bleibt Ihre Rolle bis zu einer Stunde lang für die Verbindung autorisiert.

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

PostgreSQL verwendet das Rollenkonzept zur Verwaltung von Datenbankzugriffsberechtigungen. Eine Rolle kann entweder als Datenbankbenutzer oder als Gruppe von Datenbankbenutzern

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

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

Authentifizierung

Stellen Sie mit der `admin`-Rolle eine Verbindung mit Ihrem Cluster her. Nachdem Sie Ihre Datenbank verbunden haben, verwenden Sie den Befehl `AWS IAM GRANT`, um eine benutzerdefinierte Datenbankrolle mit der IAM-Identität zu verknüpfen, die berechtigt ist, sich mit dem Cluster zu verbinden – wie im folgenden Beispiel gezeigt.

```
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 für die Verbindungsherstellung zu Ihrem Cluster.

Autorisierung

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

IAM-Richtlinienaktionen in Aurora DSQL verwenden

Welche IAM-Richtlinienaktion Sie verwenden, hängt von der Rolle ab, die Sie für die Verbindung mit Ihrem Cluster nutzen: 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 zur Verbindung mit Clustern

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

```
"dsql:DbConnectAdmin"
```

Wenn Sie mit einer benutzerdefinierten Datenbankrolle eine Verbindung zu Ihrem Cluster herstellen, müssen Sie zuerst die IAM-Rolle der Datenbankrolle zuordnen. Die IAM-Identität, die Sie für die Verbindung mit Ihrem Cluster verwenden, muss über eine entsprechende Autorisierung verfügen, 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 Clusterverwaltung

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

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 Clusterverwaltung.

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 mithilfe von IAM und PostgreSQL

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

Widerrufen der Administratorautorisierung für die Verbindung zu Clustern

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

Nachdem die Verbindungsautorisierung für die IAM-Identität widerrufen wurde, 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 Dauer von Verbindungen finden Sie unter Kontingente und -Einschränkungen.

Autorisierungwiderruf für benutzerdefinierte Rollen zum Herstellen einer Verbindung zu Clustern

Um den Zugriff auf Datenbankrollen außer der Rolle `admin` zu widerrufen, entziehen Sie der IAM-Identität den Zugriff auf `dsql:DbConnect`. Bearbeiten Sie entweder die IAM-Richtlinie oder lösen Sie die Richtlinie von der Identität.

Sie können die Verknüpfung zwischen der Datenbankrolle und IAM auch entfernen, indem Sie den Befehl `AWS IAM REVOKE` in Ihrer Datenbank ausführen. Weitere Informationen zum Zugriffswiderruf für Datenbankrollen finden Sie unter Widerrufen der Datenbankautorisierung einer IAM-Rolle.

Die Berechtigungen der vordefinierten `admin`-Datenbankrolle können nicht verwaltet werden. Informationen zur Berechtigungsverwaltung von benutzerdefinierten Datenbankrollen finden Sie unter PostgreSQL-Berechtigungen. 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 AWS CLI, läuft das oder SDKs, das Token standardmäßig automatisch in 15 Minuten ab. Die maximale Dauer beträgt 604.800 Sekunden, also eine Woche. Um von Ihrem Client aus erneut eine Verbindung zu Aurora DSQL herzustellen, können Sie dasselbe Authentifizierungstoken verwenden, sofern es nicht abgelaufen ist, oder Sie können ein neues Token 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 AWS CLI, AWS SDKs um ein Token zu generieren.

Je nachdem, welche Datenbankrolle Sie für die Verbindung verwenden, benötigen Sie mindestens die unter Verbindungsaufbau mit Ihrem Cluster mithilfe von IAM 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 über die Konsole erstellen.

Generieren eines Authentifizierungstokens

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Aurora DSQL-Konsole unterhttps://console.aws.amazon.com/dsql.
2. Wählen Sie die Cluster-ID des Clusters, für den Sie ein Authentifizierungstoken erstellen möchten. Wenn Sie noch keinen Cluster erstellt haben, folgen Sie den Schritten unter Schritt 1: Erstellen eines Einzelregions-Clusters mit Aurora DSQL oder Schritt 4 (optional): Erstellen Sie einen Cluster mit mehreren Regionen.
3. Wählen Sie Verbinden und dann Token abrufen aus.
4. Wählen Sie aus, ob Sie die Verbindung als `admin` oder mit einer benutzerdefinierten Datenbankrolle herstellen möchten.
5. Kopieren Sie das generierte Authentifizierungstoken und verwenden Sie es für Greifen Sie mithilfe von SQL-Clients auf Aurora DSQL zu.

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-Managementkonsole 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 die Verbindung nicht als `admin` herstellen, verwenden Sie stattdessen `generate-db-connect-auth-token`.

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

   Falls 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, um mithilfe von `psql` eine Verbindung zu Ihrem Cluster herzustellen.

   ```
   PGSSLMODE=require \
   psql --dbname postgres \
     --username admin \
     --host cluster_endpoint
   ```
5. Sie sollten nun einen Prompt zur Passworteingab sehen. Kopieren Sie das von Ihnen generierte Token und stellen Sie sicher, dass Sie keine zusätzlichen Leerzeichen oder Zeichen mitkopieren. Fügen Sie es in den folgenden `psql`-Prompt ein.

   ```
   Password for user admin:
   ```
6. Drücken Sie die Eingabetaste. Sie sollten nun einen PostgreSQL-Prompt sehen.

```
postgres=>
```

Wenn Sie die Fehlermeldung „Zugriff verweigert“ erhalten, stellen Sie sicher, dass Ihre IAM-Identität über die entsprechende `dsql:DbConnectAdmin`-Berechtigung verfügt. 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 `ACTIVE` ist, können Sie mit dem `aws dsql`-Befehl ein Authentifizierungstoken auf der CLI generieren. Sie können eine der folgenden Techniken verwenden:

> **Note**
>
> Die Token-Generierung ist ein lokaler Vorgang, bei dem die Anfrage mit Ihren aktuellen IAM-Anmeldeinformationen signiert wird. Es wird kein Kontakt hergestellt AWS , um die Anmeldeinformationen zu überprüfen. Wenn Ihre Anmeldeinformationen abgelaufen oder ungültig sind, ist die Token-Generierung trotzdem erfolgreich, aber der Verbindungsversuch schlägt fehl. Stellen Sie sicher, dass Ihre IAM-Anmeldeinformationen gültig sind, bevor Sie ein Token generieren.

- 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 nachstehenden Attribute verwendet, um ein Authentifizierungstoken für die `admin`-Rolle zu generieren.

- *your_cluster_endpoint*— Der Endpunkt des Clusters. Das Attribut folgt dem Format *your_cluster_identifier*.dsql.*region*.on.aws wie im Beispiel 01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws gezeigt.
- *region*— Der AWS-Region, wie us-east-2 oderus-east-1.

In den folgenden Beispielen wird die Ablaufzeit für das Token auf 3600 Sekunden (1 Stunde) festgelegt.

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 Status ACTIVE befindet. In den SDK-Beispielen werden die folgenden Attribute verwendet, 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 Beispiel 01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws.
- *region*(oder*RegionEndpoint*) — Der, AWS-Region in dem sich Ihr Cluster befindet, z. B. us-east-2 oderus-east-1.

Python SDK

> **Tip**
>
> AWS empfiehlt die Verwendung vonAurora DSQL-Konnektor für Python, die die Token-Generierung automatisch durchführt.

Sie können das Token wie folgt erstellen:

- 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`.

```
import boto3

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 wie folgt erstellen:

- 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) {
    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;
    return token;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    // Replace with your cluster endpoint and region
    std::string token = generateToken("your_cluster_endpoint.dsql.us-east-1.on.aws",
 "us-east-1");
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript SDK

> **Tip**
>
> AWS empfiehlt die Verwendung vonAurora DSQL-Konnektoren für Node.js, die die Token-Generierung automatisch übernehmen.

Sie können das Token wie folgt erstellen:

- 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

> **Tip**
>
> AWS empfiehlt die Verwendung vonVerbindung zu Aurora DSQL-Clustern mit einem JDBC-Konnektor herstellen, die die Token-Generierung automatisch durchführt.

Sie können das Token wie folgt erstellen:

- 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.builder().build())
                .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 wie folgt erstellen:

- 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

> **Tip**
>
> AWS empfiehlt die Verwendung von Verbindung zu Aurora DSQL-Clustern mit einem Ruby-Konnektor herstellen, die die Token-Generierung automatisch durchführt.

Sie können das Token wie folgt erstellen:

- 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::CredentialProviderChain.new.resolve

  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
  })
end
```

PHP SDK

Sie können das Token wie folgt erstellen:

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

```
<?php
require 'vendor/autoload.php';

use Aws\DSQL\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

function generateToken(string $yourClusterEndpoint, string $region): string {
    $provider = CredentialProvider::defaultProvider();
    $generator = new AuthTokenGenerator($provider);

    // Use generateDbConnectAuthToken if you are not connecting as admin
    $token = $generator->generateDbConnectAdminAuthToken($yourClusterEndpoint,
 $region);

    echo $token . PHP_EOL;
    return $token;
}
```

.NET

> **Tip**
>
> AWS empfiehlt die Verwendung von Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem .NET-Connector, die die Token-Generierung automatisch durchführt.

> **Note**
>
> Das offizielle SDK für .NET enthält keinen integrierten API-Aufruf zur Generierung eines Authentifizierungstokens für Aurora DSQL. Stattdessen müssen Sie die Utility-Klasse `DSQLAuthTokenGenerator` verwenden. Im folgenden Codebeispiel sehen Sie, wie das Authentifizierungstoken für .NET generiert wird.

Sie können das Token wie folgt erstellen:

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

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

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

// Use `DSQLAuthTokenGenerator.GenerateDbConnectAuthToken` if you are _not_ logging
 in as `admin` user
var token =
 DSQLAuthTokenGenerator.GenerateDbConnectAdminAuthToken(RegionEndpoint.USEast1,
 yourClusterEndpoint);

Console.WriteLine(token);
```

Go

> **Tip**
>
> AWS empfiehlt die Verwendung vonVerbindung zu Aurora DSQL-Clustern mit einem Go-Connector herstellen, die die Token-Generierung automatisch durchführt.

Das AWS SDK for Go v2 bietet eine integrierte Methode zum Generieren von Authentifizierungstoken im `github.com/aws/aws-sdk-go-v2/feature/dsql/auth`Paket.

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

```
package main

import (
 "context"
 "fmt"

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

func main() {
 ctx := context.Background()

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("region"))
 if err != nil {
  panic(err)
 }

 // Use auth.GenerateDbConnectAuthToken for non-admin users
 token, err := auth.GenerateDBConnectAdminAuthToken(ctx, "yourClusterEndpoint",
 "region", cfg.Credentials)
 if err != nil {
  panic(err)
 }

 fmt.Println(token)
}
```

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 auf sie zuzugreifen.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt, aber keiner bestimmten Person zugeordnet ist. Das Verwenden von IAM-Rollen stellt temporäre Sicherheitsanmeldeinformationen bereit. Sie können wie folgt vorübergehend eine IAM-Rolle annehmen:

- Durch den Rollenwechsel in der AWS-Managementkonsole
- 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 für die Verwendung von Rollen finden Sie unter IAM-Identitäten im IAM-Benutzerhandbuch.

IAM-Benutzer

Ein IAM-Benutzer ist eine Identität innerhalb von Ihnen 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 für die Verbindungsherstellung zu Ihrem Cluster

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

Die IAM-Richtlinie muss eine Berechtigung für den Zugriff auf die Clusterressourcen beinhalten. Verwenden Sie einen Platzhalter (*) oder folgen Sie den Anleitungen unter Verwenden von IAM-Bedingungsschlüsseln mit Amazon Aurora DSQL.

Autorisieren von Datenbankrollen zur Verwendung von SQL in Ihrer Datenbank

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

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

   Verwenden Sie die `admin`-Datenbankrolle mit einer IAM-Identität, die für die IAM-Aktion `dsql:DbConnectAdmin` autorisiert ist, um eine Verbindung 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-Rollen-ARN zu.

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

4. Der Datenbankrolle Berechtigungen auf Datenbankebene erteilen

   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-Berechtigungen in der PostgreSQL-Dokumentation.

Anzeigen der Zuordnungen von IAM- zu Datenbankrollen

Zur Anzeige der Zuordnungen von IAM-Rollen zu Datenbankrollen führen Sie eine Abfrage in der `sys.iam_pg_role_mappings`-Systemtabelle durch.

```
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)
```

In dieser Tabelle werden alle Zuordnungen zwischen IAM-Rollen (identifiziert durch ihren ARN) und PostgreSQL-Datenbankrollen angegeben.

Widerrufen der Datenbankautorisierung einer IAM-Rolle

Sie verwenden die `AWS IAM REVOKE`-Operation, um eine Datenbankautorisierung zu widerrufen.

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

Weitere Informationen zum Autorisierungswiderruf finden Sie unter Widerrufen der Autorisierung mithilfe von 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 Typensystem.

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 sind dokumentiert.

Aurora DSQL bietet auch erweiterte Funktionen wie optimistische Parallelitätssteuerung und verteilte Schemaverwaltung. Mit diesen Funktionen können Sie die vertrauten Tools von PostgreSQL verwenden und gleichzeitig von der Leistung und Skalierbarkeit moderner, cloudnativer, verteilter Anwendungen profitieren.

PostgreSQL-Kompatibilitätsmerkmale

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

Wire-Protokoll

Aurora DSQL verwendet das standardmäßige Wire-Protokoll in PostgreSQL V3. Dies ermöglicht die Integration mit standardmäßigen PostgreSQL-Clients, -Treibern und -Tools. Aurora SQL ist beispielsweise kompatibel mit `psql`, `pgjdbc` und `psycopg`.

SQL-Syntax

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

- Umgang mit Nullwerten
- Verhalten der Sortierreihenfolge
- Skalierung und Genauigkeit bei numerischen Operationen
- Äquivalenz für Zeichenkettenoperationen

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

Transaktionsmanagement

Aurora DSQL bewahrt die zentralen Eigenschaften von PostgreSQL, darunter ACID-konforme Transaktionen und eine Isolationsebene, die dem PostgreSQL-Modus „Repeatable Read“ entspricht. Weitere Informationen finden Sie unter Parallelitätssteuerung in Aurora DSQL.

Vorteile einer verteilten Architektur

Das verteilte Shared-Nothing-Design von Aurora DSQL bietet Leistungs- und Skalierbarkeitsvorteile im Vergleich zu herkömmlichen Einzelknoten-Datenbanken. Zu den wichtigsten Funktionen gehören die folgenden:

Optimistic Concurrency Control (OCC)

Aurora DSQL verwendet ein optimistisches Parallelitätskontrollmodell. Dieser sperrenfreie Ansatz verhindert, dass sich Transaktionen gegenseitig blockieren, beseitigt Deadlocks und ermöglicht eine parallele Ausführung mit hohem Durchsatz. Diese Features machen Aurora DSQL besonders wertvoll für Anwendungen, die eine gleichbleibende Leistung im großen Maßstab erfordern. Weitere Beispiele finden Sie unter Parallelitätssteuerung in Aurora DSQL.

Asynchrone DDL-Operationen

Aurora DSQL führt DDL-Operationen asynchron aus, was unterbrechungsfreie Lese- und Schreibvorgänge bei Schemaänderungen ermöglicht. Die verteilte Architektur ermöglicht es Aurora DSQL, die folgenden Aktionen auszuführen:

- Ausführen von DDL-Operationen als Hintergrundaufgaben, um Unterbrechungen zu minimieren.
- Koordinierung von Katalogänderungen als hochkonsistente, verteilte Transaktionen. Dies gewährleistet atomare Sichtbarkeit über alle Knoten hinweg – selbst bei Ausfällen oder gleichzeitigen Operationen.
- Aurora DSQL-Einsatz in einem vollständig verteilten, führerlosen Modus über mehrere Availability Zones, mit entkoppelten Compute- und Speicherebenen.

Weitere Informationen zur Verwendung des EXPLAIN-Befehls in PostgreSQL finden Sie unter. DDL und verteilte Transaktionen in Aurora DSQL

Kompatibilität der SQL-Features in Aurora DSQL

In den folgenden Abschnitten erfahren Sie mehr über Aurora DSQL-Unterstützung für PostgreSQL-Datentypen und SQL-Befehle.

Topics

- Unterstützte Datentypen in Aurora DSQL
- Unterstütztes SQL für Aurora DSQL
- Unterstützte Teilmengen von PostgreSQL-Befehlen in Aurora DSQL
- Migration von PostgreSQL zu Aurora DSQL

Unterstützte Datentypen in Aurora DSQL

Aurora DSQL unterstützt eine Teilmenge der gängigen PostgreSQL-Typen.

Topics

- Numerische Datentypen
- Zeichendatentypen
- Datums- und Uhrzeit-Datentypen
- Sonstige Datentypen
- Datentypen zur Abfragelaufzeit

Numerische Datentypen

Aurora DSQL unterstützt die folgenden numerischen PostgreSQL-Datentypen.

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

Name	Aliase	Reichweite und Präzision	Speichergröße	Index-Unterstützung
`bigint`	`int8`	-9223372036854775808 bis +9223372036854775807	8 Bytes	Ja
`real`	`float4`	Genauigkeit von 6 Dezimalstellen	4 Bytes	Ja
`double precision`	`float8`	Genauigkeit von 15 Dezimalstellen	8 Bytes	Ja

`numeric [ (p, s) ]`	`decimal [ (p, s) ]`
	
	`dec[ (p,s)]`
		Exakte numerische Werte mit wählbarer Genauigkeit. Die maximale Genauigkeit für alle DECIMAL-Werte beträgt 38, und es sind maximal 37 Dezimalstellen zulässig.[1] Der Standardwert ist `numeric (18,6)`.	8 Byte + 2 Byte pro Präzisionsstelle. Die maximale Größe ist 27 Byte.	Ja

[1] – wenn Sie bei der Ausführung von `CREATE TABLE` oder `ALTER TABLE ADD COLUMN` keine explizite Größe angeben, wird Aurora DSQL die Standardwerte durchsetzen. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE`- oder `INSERT`-Anweisungen ausführen.

Zeichendatentypen

Aurora DSQL unterstützt die folgenden PostgreSQL-Zeichendatentypen.

Name	Aliase	Description	Aurora DSQL-Beschränkung	Speichergröße	Index-Unterstützung
`character [ (n) ]`	`char [ (n) ]`	Zeichenfolge mit fester Länge	4096 Byte[1]	Variabel bis zu 4100 Byte	Ja

Name	Aliase	Description	Aurora DSQL-Beschränkung	Speichergröße	Index-Unterstützung
`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 nachgestellte Leerzeichen semantisch keine Bedeutung haben.	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 `ALTER TABLE ADD COLUMN` keine explizite Größe angeben, wird Aurora DSQL die Standardwerte durchsetzen. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE`- oder `INSERT`-Anweisungen ausführen.

Datums- und Uhrzeit-Datentypen

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

Name	Aliase	Description	Bereich	Behebung	Speichergröße	Index-Unterstützung
`date`		Kalenderdatum (Jahr,	4 713 BC – 5 874 897 AD	1 Tag	4 Bytes	Ja

Name	Aliase	Description	Bereich	Behebung	Speichgröße	Index-Unterstützung
		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 BC – 294276 AD	1 Mikrosekunde	8 Bytes	Ja
`timestamp[(p)]with time zone`	`timestz`	Datum und Uhrzeit, einschließlich Zeitzone	4713 BC – 294276 AD	1 Mikrosekunde	8 Bytes	Ja
`interval[fields][(p)]`		Zeitspanne	-178 000 000 Jahre – 178 000 000 Jahre	1 Mikrosekunde	16 Byte	Nein

Sonstige Datentypen

Aurora DSQL unterstützt die folgenden sonstigen PostgreSQL-Datentypen.

Name	Aliase	Description	Aurora DSQL-Beschränkung	Speichergröße	Index-Unterstützung
`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 eindeutige ID		16 Byte	Ja

[1] – wenn Sie bei der Ausführung von `CREATE TABLE` oder `ALTER TABLE ADD COLUMN` keine explizite Größe angeben, wird Aurora DSQL die Standardwerte durchsetzen. Aurora DSQL wendet Grenzwerte an, wenn Sie `UPDATE`- oder `INSERT`-Anweisungen ausführen.

Datentypen zur Abfragelaufzeit

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 unterstützt:

Array-Typ

> Aurora DSQL unterstützt Arrays der unterstützten Datentypen. Sie können z. B. ein Array von Ganzzahlen verwenden. 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 während der Abfrageausführung Arrays in Ausdrücken, Funktionsausgaben oder temporären Berechnungen verwenden.

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

Die Funktion gibt dann eine Antwort wie folgt zurück:

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

inet-Typ

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

JSON-Laufzeitfunktionen

Aurora DSQL unterstützt JSON und JSONB als Laufzeitdatentypen für die Abfrageverarbeitung. Speichern Sie JSON-Daten als `text` Spalten und wandeln Sie sie während der Abfrageausführung in JSON um, um PostgreSQL-JSON-Funktionen und -Operatoren zu verwenden.

Aurora DSQL unterstützt die meisten PostgreSQL-JSON-Funktionen aus Abschnitt 9.1.6 JSON-Funktionen und -Operatoren mit identischem Verhalten.

Funktionen, die JSON- oder JSONB-Typen zurückgeben, erfordern möglicherweise eine zusätzliche Typumwandlung in `text`, um korrekt dargestellt zu werden.

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

Die Funktion gibt eine Antwort ähnlich wie folgt zurück:

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

Unterstütztes SQL für Aurora DSQL

Aurora DSQL unterstützt eine Vielzahl von wichtigen PostgreSQL-SQL-Features. In den folgenden Abschnitten erfahren Sie mehr über die Unterstützung von PostgreSQL-Ausdrücken im Allgemeinen. Diese Liste ist nicht umfassend.

SELECT command

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

Primär-Klausel	Unterstützte Klauseln
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	
`WITH` (allgemeine Tabellena usdrücke)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`

Primär-Klausel	Unterstützte Klauseln
OVER	RANK (), PARTITION BY
FOR UPDATE	

Data Definition Language (DDL)

Aurora DSQL unterstützt die folgenden PostgreSQL-DDL-Befehle.

Befehl	Primär-Klausel	Unterstützte Klauseln
CREATE	TABLE	Informationen zur unterstützten Syntax des CREATE TABLE-Befehls finden Sie unter CREATE TABLE.
ALTER	TABLE	Informationen zur unterstützten Syntax des ALTER TABLE-Befehls finden Sie unter ALTER TABLE.
DROP	TABLE	
CREATE	[UNIQUE] INDEX ASYNC	Sie können diesen Befehl mit den folgenden Parametern verwenden: ON, NULLS FIRST, NULLS LAST. Informationen zur unterstützten Syntax des CREATE INDEX ASYNC-Befehls finden Sie unter Asynchrone Indizes in Aurora DSQL.
DROP	INDEX	
CREATE	VIEW	Weitere Informationen zur unterstützten Syntax des CREATE VIEW-Befehls finden Sie unter CREATE VIEW.

Befehl	Primär-Klausel	Unterstützte Klauseln
ALTER	VIEW	Informationen zur unterstützten Syntax des ALTER VIEW-Befehls finden Sie unter ALTER VIEW.
DROP	VIEW	Informationen zur unterstützten Syntax des DROP VIEW-Befehls finden Sie unter DROP VIEW.
CREATE	SEQUENCE	Informationen zur unterstützten Syntax des CREATE SEQUENCE-Befehls finden Sie unter CREATE SEQUENCE.
ALTER	SEQUENCE	Informationen zur unterstützten Syntax des ALTER SEQUENCE-Befehls finden Sie unter ALTER SEQUENCE.
DROP	SEQUENCE	Informationen zur unterstützten Syntax des DROP SEQUENCE-Befehls finden Sie unter DROP SEQUENCE.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Data Manipulation Language (DML)

Aurora DSQL unterstützt die folgenden PostgreSQL-DML-Befehle.

Befehl	Primär-Klausel	Unterstützte Klauseln
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT)

Befehl	Primär-Klausel	Unterstützte Klauseln
		`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`

Transaction Control Language (TCL)

Aurora DSQL unterstützt die folgenden PostgreSQL-TCL-Befehle.

Befehl	Unterstützte Klauseln	Alias
`COMMIT`	`[WORK | TRANSACTION ]` `[AND NO CHAIN]`	`END`

`BEGIN`	`[WORK | TRANSACTION ]`
	`[ISOLATION LEVEL REPEATABLE READ]`
	`[READ WRITE | READ ONLY]`	

`START TRANSACTION`	`[ISOLATION LEVEL REPEATABLE READ]` `[READ WRITE | READ ONLY]`	
`ROLLBACK`	`[WORK | TRANSACTION ]`	`ABORT`

Befehl	Unterstützte Klauseln	Alias
	[AND NO CHAIN]	

Dienstprogrammbefehle

Aurora DSQL unterstützt die folgenden PostgreSQL-Dienstprogrammbefehle:

- `EXPLAIN`
- `ANALYZE` (nur der Name der Beziehung)

Unterstützte Teilmengen von PostgreSQL-Befehlen in Aurora DSQL

Dieser Abschnitt enthält detaillierte Informationen zu unterstützten SQL-Befehlen, wobei der Schwerpunkt auf Befehlen mit umfangreichen Parametersätzen und Unterbefehlen liegt. CREATE TABLE in PostgreSQL bietet beispielsweise viele Klauseln und Parameter, von denen eine Teilmenge von Aurora DSQL unterstützt wird. In diesem Abschnitt werden die unterstützten Teilmengen gängiger SQL-Befehle unter Verwendung vertrauter PostgreSQL-Syntaxelemente beschrieben, die Aurora DSQL unterstützt.

Topics

- CREATE TABLE
- ALTER TABLE
- CREATE SEQUENCE
- ALTER SEQUENCE
- DROP SEQUENCE
- CREATE VIEW
- ALTER VIEW
- DROP VIEW

`CREATE TABLE`

`CREATE TABLE` definiert 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 |
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options ) |
  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 [, ... ] ) ]
```

Identitätsspalten

> **Note**
>
> Bei der Verwendung von Identitätsspalten sollte der Cache-Wert sorgfältig abgewogen werden. Weitere Informationen finden Sie in der Erläuterung „Wichtig“ auf der CREATE SEQUENCE Seite.
> Hinweise zur optimalen Verwendung von Identitätsspalten auf der Grundlage von Workload-Mustern finden Sie unterArbeiten mit Sequenzen und Identitätsspalten.

Die `GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options )` Klausel erstellt die Spalte als Identitätsspalte. Ihr wird eine implizite Sequenz angehängt, und in neu eingefügten Zeilen enthält die Spalte automatisch Werte aus der ihr zugewiesenen Sequenz. Eine solche Spalte ist implizit. `NOT NULL`

Die Klauseln `ALWAYS` und `BY DEFAULT` bestimmen, wie explizit vom Benutzer angegebene Werte in `INSERT` Befehlen und Befehlen behandelt werden. `UPDATE`

Wenn in einem `INSERT` Befehl ausgewählt `ALWAYS` ist, wird ein benutzerdefinierter Wert nur akzeptiert, wenn in der `INSERT` Anweisung Folgendes angegeben ist. `OVERRIDING SYSTEM VALUE` Wenn ausgewählt `BY DEFAULT` ist, hat der benutzerdefinierte Wert Vorrang.

Wenn in einem `UPDATE` Befehl diese Option ausgewählt `ALWAYS` ist, wird jede Aktualisierung der Spalte auf einen anderen Wert als diesen `DEFAULT` abgelehnt. Wenn diese Option ausgewählt `BY DEFAULT` ist, kann die Spalte normal aktualisiert werden. (Es gibt keine `OVERRIDING` Klausel für den `UPDATE` Befehl.)

Die *`sequence_options`* Klausel kann verwendet werden, um die Parameter der Sequenz zu überschreiben. Zu den verfügbaren Optionen gehören die für`CREATE SEQUENCE`, plus`SEQUENCE NAME` *`name`*. Ohne `SEQUENCE NAME` diese Option wählt das System einen unbenutzten Namen für die Sequenz.

`ALTER TABLE`

`ALTER TABLE` ändert 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
    ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET
 sequence_option | RESTART [ [ WITH ] restart ] } [...]
```

```
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
```

Aktionen in der Spalte „Identität“

SET GENERATED { ALWAYS | BY DEFAULT } / SET *sequence_option* / RESTART

Diese Formulare ändern, ob es sich bei einer Spalte um eine Identitätsspalte handelt, oder ändern das Generierungsattribut einer vorhandenen Identitätsspalte. Details dazu finden Sie unter CREATE TABLE. SET DEFAULTEbenso wirken sich diese Formulare nur auf das Verhalten der nachfolgenden UPDATE Befehle INSERT und aus; sie bewirken nicht, dass Zeilen, die sich bereits in der Tabelle befinden, geändert werden.

Das *sequence_option* ist eine Option, die von ALTER SEQUENCE z. B. unterstützt wirdINCREMENT BY. Diese Formulare ändern die Reihenfolge, die einer vorhandenen Identitätsspalte zugrunde liegt.

DROP IDENTITY [IF EXISTS]

Dieses Formular entfernt die Identitätseigenschaft aus einer Spalte. Wenn angegeben DROP IDENTITY IF EXISTS ist und es sich bei der Spalte nicht um eine Identitätsspalte handelt, wird kein Fehler ausgegeben. In diesem Fall wird stattdessen eine Mitteilung herausgegeben.

CREATE SEQUENCE

CREATE SEQUENCE— definiere einen neuen Sequenzgenerator.

> **Important**
>
> In PostgreSQL CACHE ist die Angabe optional und der Standardwert ist 1. In einem verteilten System wie Amazon Aurora DSQL erfordern Sequenzoperationen Koordination, und eine Cachegröße von 1 kann den Koordinationsaufwand bei hoher Parallelität erhöhen. Größere Cache-Werte ermöglichen es zwar, Sequenznummern aus lokal vorab zugewiesenen Bereichen bereitzustellen, was den Durchsatz verbessert, aber ungenutzte reservierte Werte können verloren gehen, wodurch Lücken und Reihenfolgen sichtbarer werden. Da sich Anwendungen in ihrer Sensitivität in Bezug auf die Reihenfolge der Zuweisungen im Vergleich zum Durchsatz unterscheiden, muss CACHE Amazon Aurora DSQL explizit spezifiziert werden und unterstützt derzeit CACHE = 1 oderCACHE >= 65536, wobei eine klare Unterscheidung zwischen einem Zuweisungsverhalten, das eher der strikt sequentiellen

Generierung ähnelt, und einer Zuweisung, die für sehr gleichzeitige Workloads optimiert ist, gewährleistet wird.
Wenn Sequenzwerte garantiert eindeutig sind`CACHE >= 65536`, sie aber möglicherweise nicht in streng aufsteigender Reihenfolge über Sitzungen hinweg generiert werden, kann es zu Lücken kommen, insbesondere wenn zwischengespeicherte Werte nicht vollständig verbraucht werden. Diese Eigenschaften stimmen mit der PostgreSQL-Semantik für zwischengespeicherte Sequenzen bei gleichzeitiger Verwendung überein, wobei beide Systeme unterschiedliche Werte garantieren, aber keine strikte sequentielle Reihenfolge zwischen den Sitzungen garantieren.
Innerhalb einer einzelnen Clientsitzung erscheinen Sequenzwerte möglicherweise nicht immer strikt ansteigend, insbesondere außerhalb expliziter Transaktionen. Dieses Verhalten ähnelt PostgreSQL-Bereitstellungen, die Verbindungspooling verwenden. Ein Zuweisungsverhalten, das eher einer PostgreSQL-Umgebung mit einer Sitzung ähnelt, kann erreicht werden, indem Sequenzwerte verwendet `CACHE = 1` oder innerhalb expliziter Transaktionen abgerufen werden.
Mit folgt die Sequenzzuweisung dem Sequenzverhalten von PostgreSQL`CACHE = 1`, das nicht zwischengespeichert wurde.
Hinweise zur optimalen Verwendung von Sequenzen, die auf Workload-Mustern basieren, finden Sie unter. Arbeiten mit Sequenzen und Identitätsspalten

Unterstützte Syntax

```
CREATE SEQUENCE [ IF NOT EXISTS ] name CACHE cache
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ OWNED BY { table_name.column_name | NONE } ]

where data_type is BIGINT
      and cache = 1 or cache >= 65536
```

Description

`CREATE SEQUENCE`erstellt einen neuen Sequenznummerngenerator. Dies beinhaltet das Erstellen und Initialisieren einer neuen speziellen einzeiligen Tabelle mit dem Namen. *`name`* Der Generator wird dem Benutzer gehören, der den Befehl ausgibt.

Wenn ein Schemaname angegeben wird, wird die Sequenz im angegebenen Schema erstellt. Andernfalls wird sie im aktuellen Schema erstellt. Der Sequenzname muss sich vom Namen jeder anderen Relation (Tabelle, Sequenz, Index, Ansicht, materialisierte Ansicht oder Fremdtabelle) im selben Schema unterscheiden.

Nachdem eine Sequenz erstellt wurde, verwenden Sie die Funktionen`nextval`, und`currval`, `setval` um die Sequenz zu bearbeiten. Diese Funktionen sind in dokumentiertSequenz-Bearbeitungsfunktionen.

Sie können eine Sequenz zwar nicht direkt aktualisieren, aber Sie können eine Abfrage wie die folgende verwenden:

```
SELECT * FROM name;
```

um einige Parameter und den aktuellen Status einer Sequenz zu untersuchen. Insbesondere zeigt das `last_value` Feld der Sequenz den letzten Wert an, der von einer Sitzung zugewiesen wurde. (Natürlich kann dieser Wert zum Zeitpunkt der Ausgabe veraltet sein, wenn andere Sitzungen aktiv `nextval` Aufrufe ausführen.) Andere Parameter wie *`increment`* und *`maxvalue`* können in der `pg_sequences` Ansicht beobachtet werden.

Parameters

`IF NOT EXISTS`

Geben Sie keinen Fehler aus, wenn bereits eine Relation mit demselben Namen existiert. In diesem Fall wird eine Mitteilung ausgegeben. Beachten Sie, dass es keine Garantie dafür gibt, dass die bestehende Beziehung der Sequenz ähnelt, die erstellt worden wäre — es ist möglicherweise nicht einmal eine Sequenz.

`name`

Der Name (optional schemaqualifiziert) der zu erstellenden Sequenz.

`data_type`

Die optionale Klausel `AS` *`data_type`* gibt den Datentyp der Sequenz an. Gültige Typen sind`bigint`. `bigint`ist die Standardeinstellung. Der Datentyp bestimmt die standardmäßigen Minimal- und Maximalwerte der Sequenz.

increment

Die optionale Klausel `INCREMENT BY` *`increment`* gibt an, welcher Wert zum aktuellen Sequenzwert hinzugefügt wird, um einen neuen Wert zu erstellen. Ein positiver Wert ergibt eine aufsteigende Sequenz, ein negativer Wert eine absteigende Sequenz. Der Standardwert ist 1.

minvalue / **`NO MINVALUE`**

Die optionale Klausel `MINVALUE` *`minvalue`* bestimmt den Mindestwert, den eine Sequenz generieren kann. Wenn diese Klausel nicht angegeben oder angegeben `NO MINVALUE` wird, werden Standardwerte verwendet. Die Standardeinstellung für eine aufsteigende Sequenz ist 1. Die Standardeinstellung für eine absteigende Sequenz ist der Mindestwert des Datentyps.

maxvalue / **`NO MAXVALUE`**

Die optionale Klausel `MAXVALUE` *`maxvalue`* bestimmt den Höchstwert für die Sequenz. Wenn diese Klausel nicht angegeben oder angegeben `NO MAXVALUE` wird, werden Standardwerte verwendet. Die Standardeinstellung für eine aufsteigende Sequenz ist der Maximalwert des Datentyps. Die Standardeinstellung für eine absteigende Sequenz ist -1.

`CYCLE` / **`NO CYCLE`**

Die `CYCLE` Option ermöglicht es, dass die Sequenz umgebrochen wird, wenn das *`maxvalue`* oder *`minvalue`* durch eine aufsteigende bzw. absteigende Sequenz erreicht wurde. Wenn das Limit erreicht ist, wird als nächste Zahl das *`minvalue`* Oder *`maxvalue`* generiert.

Wenn `NO CYCLE` angegeben, geben alle Aufrufe von, `nextval` nachdem die Sequenz ihren Maximalwert erreicht hat, einen Fehler zurück. Wenn keiner `CYCLE` von beiden `NO CYCLE` angegeben `NO CYCLE` ist, ist dies die Standardeinstellung.

start

Die optionale Klausel `START WITH` *`start`* ermöglicht es, dass die Sequenz an einer beliebigen Stelle beginnt. Der Standardstartwert gilt *`minvalue`* für aufsteigende und *`maxvalue`* für absteigende Sequenzen.

cache

Die Klausel `CACHE` *`cache`* gibt an, wie viele Sequenznummern vorab zugewiesen und für einen schnelleren Zugriff im Speicher gespeichert werden sollen. Die akzeptablen Werte für `CACHE` in Aurora DSQL sind 1 oder eine beliebige Zahl >= 65536. Der Mindestwert ist 1 (es kann jeweils nur ein Wert generiert werden, d. h. kein Cache).

OWNED BY *table_name*.*column_name* / OWNED BY NONE

Die OWNED BY Option bewirkt, dass die Sequenz einer bestimmten Tabellenspalte zugeordnet wird, sodass, wenn diese Spalte (oder ihre gesamte Tabelle) gelöscht wird, auch die Sequenz automatisch gelöscht wird. Die angegebene Tabelle muss denselben Besitzer haben und sich im selben Schema wie die Sequenz befinden. OWNED BY NONE, die Standardeinstellung, gibt an, dass es keine solche Zuordnung gibt.

Hinweise

Wird verwendetDROP SEQUENCE, um eine Sequenz zu entfernen.

Sequenzen basieren auf bigint Arithmetik, sodass der Bereich den Bereich einer 8-Byte-Ganzzahl (-9223372036854775808 bis 9223372036854775807) nicht überschreiten darf.

Da Aufrufe niemals rückgängig gemacht werden, können Sequenzobjekte nicht verwendet werden, wenn eine „lückenlose“ nextval Zuweisung setval von Sequenznummern erforderlich ist.

In jeder Sitzung werden aufeinanderfolgende Sequenzwerte während eines Zugriffs auf das Sequenzobjekt zugewiesen und zwischengespeichert und die Werte des Sequenzobjekts entsprechend erhöht. last_value Bei den nächsten *cache* -1 Verwendungen von nextval innerhalb dieser Sitzung werden dann einfach die vorab zugewiesenen Werte zurückgegeben, ohne das Sequenzobjekt zu berühren. Somit gehen alle innerhalb einer Sitzung zugewiesenen, aber nicht verwendeten Zahlen verloren, wenn die Sitzung endet, was zu „Löchern“ in der Sequenz führt.

Darüber hinaus wird zwar garantiert, dass mehreren Sitzungen unterschiedliche Sequenzwerte zugewiesen werden, die Werte können jedoch bei der Berücksichtigung aller Sitzungen nicht in der richtigen Reihenfolge generiert werden. Bei einer *cache* Einstellung von 10 könnte Sitzung A beispielsweise die Werte 1.. 10 reservieren und nextval =1 zurückgeben, dann könnte Sitzung B die Werte 11.. 20 reservieren und =11 zurückgeben, bevor Sitzung A nextval =2 generiert hat. nextval Bei einer *cache* Einstellung von eins kann man also davon ausgehen, dass nextval Werte sequentiell generiert werden. Bei einer *cache* Einstellung größer als eins sollten Sie nur davon ausgehen, dass die nextval Werte alle unterschiedlich sind, nicht dass sie rein sequentiell generiert werden. Außerdem last_value wird der letzte Wert wiedergegeben, der für eine Sitzung reserviert wurde, unabhängig davon, ob er noch von zurückgegeben wurde oder nicht. nextval

Eine weitere Überlegung ist, dass ein in einer solchen Sequenz setval ausgeführter Vorgang von anderen Sitzungen erst bemerkt wird, wenn sie alle vorab zugewiesenen Werte aufgebraucht haben, die sie zwischengespeichert haben.

Beispiele

Erstellen Sie eine aufsteigende Sequenz mit dem Namen`serial`, beginnend bei 101:

```
CREATE SEQUENCE serial CACHE 65536 START 101;
```

Wählen Sie die nächste Zahl aus dieser Sequenz aus:

```
SELECT nextval('serial');

 nextval
---------
     101
```

Wählen Sie die nächste Zahl aus dieser Sequenz aus:

```
SELECT nextval('serial');

 nextval
---------
     102
```

Verwenden Sie diese Sequenz in einem `INSERT` Befehl:

```
INSERT INTO distributors VALUES (nextval('serial'), 'nothing');
```

Setzen Sie die Sequenz auf einen bestimmten Wert zurück, indem Sie`setval`:

```
SELECT setval('serial', 200);
SELECT nextval('serial');

 nextval
---------
     201
```

Kompatibilität

`CREATE SEQUENCE`entspricht dem SQL-Standard, mit den folgenden Ausnahmen:

- Das Abrufen des nächsten Werts erfolgt mithilfe der `nextval()` Funktion anstelle des `NEXT VALUE FOR` Standardausdrucks.

- Die `OWNED BY` Klausel ist eine PostgreSQL-Erweiterung.

`ALTER SEQUENCE`

`ALTER SEQUENCE`— die Definition eines Sequenzgenerators ändern.

> **Important**
>
> Bei der Verwendung von Sequenzen sollte der Cache-Wert sorgfältig abgewogen werden. Weitere Informationen finden Sie im Callout „Wichtig“ auf der `CREATE SEQUENCE` Seite. Hinweise zur optimalen Verwendung von Sequenzen, die auf Workload-Mustern basieren, finden Sie unterArbeiten mit Sequenzen und Identitätsspalten.

Unterstützte Syntax

```
ALTER SEQUENCE [ IF EXISTS ] name
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
    [ CACHE cache ]
    [ OWNED BY { table_name.column_name | NONE } ]
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema

where cache is 1 or cache >= 65536
```

Description

`ALTER SEQUENCE`ändert die Parameter eines vorhandenen Sequenzgenerators. Alle Parameter, die nicht ausdrücklich im `ALTER SEQUENCE` Befehl festgelegt wurden, behalten ihre vorherigen Einstellungen bei.

Sie müssen die zu verwendende Sequenz besitzen`ALTER SEQUENCE`. Um das Schema einer Sequenz 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 Sequenz verfügen. (Diese Einschränkungen legen fest, dass das Ändern des Besitzers nichts bewirkt, was Sie nicht tun könnten, indem Sie die Sequenz löschen und neu erstellen. Ein Superuser kann jedoch trotzdem den Besitz einer beliebigen Sequenz ändern.)

Parameters

name

Der Name (optional schemaqualifiziert) einer Sequenz, die geändert werden soll.

IF EXISTS

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

increment

Die Klausel INCREMENT BY *increment* ist optional. Ein positiver Wert ergibt eine aufsteigende Reihenfolge, ein negativer Wert eine absteigende Sequenz. Falls nicht angegeben, wird der alte Inkrementwert beibehalten.

minvalue / **NO MINVALUE**

Die optionale Klausel MINVALUE *minvalue* bestimmt den Mindestwert, den eine Sequenz generieren kann. Wenn NO MINVALUE angegeben, werden die Standardwerte 1 und der Mindestwert des Datentyps für aufsteigende bzw. absteigende Sequenzen verwendet. Wenn keine der Optionen angegeben ist, wird der aktuelle Mindestwert beibehalten.

maxvalue / **NO MAXVALUE**

Die optionale Klausel MAXVALUE *maxvalue* bestimmt den Maximalwert für die Sequenz. Falls NO MAXVALUE angegeben, werden die Standardwerte des Maximalwerts des Datentyps und -1 für aufsteigende bzw. absteigende Sequenzen verwendet. Wenn keine der Optionen angegeben ist, wird der aktuelle Maximalwert beibehalten.

CYCLE

Das optionale CYCLE Schlüsselwort kann verwendet werden, um zu ermöglichen, dass die Sequenz umgebrochen wird, wenn das *maxvalue* oder *minvalue* durch eine aufsteigende bzw. absteigende Sequenz erreicht wurde. Wenn das Limit erreicht ist, wird als nächste Zahl das *minvalue* Oder *maxvalue* generiert.

NO CYCLE

Wenn das optionale NO CYCLE Schlüsselwort angegeben wird, geben alle Aufrufe von, nextval nachdem die Sequenz ihren Maximalwert erreicht hat, einen Fehler zurück. Wenn keines CYCLE oder beide angegeben NO CYCLE sind, wird das alte Zyklusverhalten beibehalten.

start

Die optionale Klausel START WITH *start* ändert den aufgezeichneten Startwert der Sequenz. Dies hat keine Auswirkung auf den aktuellen Sequenzwert; es legt lediglich den Wert fest, den future ALTER SEQUENCE RESTART Befehle verwenden werden.

restart

Die optionale Klausel RESTART [WITH *restart*] ändert den aktuellen Wert der Sequenz. Dies ist vergleichbar mit dem Aufrufen der setval Funktion mit is_called =false: Der angegebene Wert wird beim nächsten Aufruf von zurückgegebennextval. Das Schreiben RESTART ohne *restart* Wert entspricht der Angabe des Startwerts, der von aufgezeichnet CREATE SEQUENCE oder zuletzt von gesetzt wurdeALTER SEQUENCE START WITH.

Im Gegensatz zu einem setval Anruf ist eine RESTART Operation in einer Sequenz transaktionaler Natur und verhindert, dass gleichzeitige Transaktionen Zahlen aus derselben Sequenz abrufen. Wenn das nicht der gewünschte Betriebsmodus ist, setval sollte er verwendet werden.

cache

Die Klausel CACHE *cache* ermöglicht es, Sequenznummern vorab zuzuweisen und für einen schnelleren Zugriff im Speicher zu speichern. Der Wert muss entweder 1 oder ein Wert >= 65536 sein. Wenn nicht angegeben, wird der alte Cache-Wert beibehalten. Weitere Informationen zum Verhalten von Caches finden Sie in den Anleitungen unterCREATE SEQUENCE.

OWNED BY *table_name*.*column_name* / OWNED BY NONE

Die OWNED BY Option bewirkt, dass die Sequenz einer bestimmten Tabellenspalte zugeordnet wird. Wenn also diese Spalte (oder ihre gesamte Tabelle) gelöscht wird, wird auch die Sequenz automatisch gelöscht. Falls angegeben, ersetzt diese Zuordnung alle zuvor angegebenen Verknüpfungen für die Sequenz. Die angegebene Tabelle muss denselben Besitzer haben und sich im selben Schema wie die Sequenz befinden. Durch die Angabe OWNED BY NONE wird jede bestehende Zuordnung entfernt, wodurch die Sequenz „eigenständig“ wird.

new_owner

Der Benutzername des neuen Besitzers der Sequenz.

new_name

Der neue Name für die Sequenz.

new_schema

Das neue Schema für die Sequenz.

Hinweise

`ALTER SEQUENCE`wirkt sich nicht sofort auf `nextval` Ergebnisse in anderen Backends als dem aktuellen aus, denen Sequenzwerte vorab zugewiesen (zwischengespeichert) sind. Sie werden alle zwischengespeicherten Werte verbrauchen, bevor sie die geänderten Parameter für die Sequenzgenerierung bemerken. Das aktuelle Backend wird sofort betroffen sein.

`ALTER SEQUENCE`hat keinen Einfluss auf den `currval` Status der Sequenz.

`ALTER SEQUENCE`kann dazu führen, dass andere Transaktionen zu OCC führen.

`ALTER TABLE`Kann aus historischen Gründen auch mit Sequenzen verwendet werden. Die einzigen Varianten`ALTER TABLE`, die bei Sequenzen zulässig sind, entsprechen den oben gezeigten Formen.

Beispiele

Starten Sie eine Sequenz neu`serial`, die bei 105 aufgerufen wird:

```
ALTER SEQUENCE serial RESTART WITH 105;
```

Kompatibilität

`ALTER SEQUENCE`entspricht dem SQL-Standard, mit Ausnahme der `SET SCHEMA` Klauseln,`AS`,,`START WITH`, und `OWNED BY` `OWNER TO``RENAME TO`, bei denen es sich um PostgreSQL-Erweiterungen handelt.

DROP SEQUENCE

`DROP SEQUENCE`— eine Sequenz entfernen.

Unterstützte Syntax

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

Description

`DROP SEQUENCE`entfernt Sequenznummerngeneratoren. Eine Sequenz kann nur von ihrem Besitzer oder einem Superuser gelöscht werden.

Parameters

`IF EXISTS`

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

`name`

Der Name (optional schemaqualifiziert) einer Sequenz.

`CASCADE`

Löscht automatisch Objekte, die von der Sequenz abhängen, und wiederum alle Objekte, die von diesen Objekten abhängen.

`RESTRICT`

Weigert sich, die Sequenz zu löschen, wenn Objekte davon abhängen. Das ist die Standardeinstellung.

Beispiele

Um die Sequenz zu entfernen`seq`:

```
DROP SEQUENCE seq;
```

Kompatibilität

`DROP SEQUENCE`entspricht dem SQL-Standard, außer dass der Standard nur das Löschen einer Sequenz pro Befehl zulässt, abgesehen von der `IF EXISTS` Option, die eine PostgreSQL-Erweiterung ist.

`CREATE VIEW`

`CREATE VIEW` definiert 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 ]
```

Description

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

`CREATE or REPLACE VIEW` ist ähnlich, wird aber ersetzt, wenn bereits eine Ansicht mit demselben Namen vorhanden ist. 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), kann jedoch zusätzliche Spalten am Ende der Liste hinzufügen. Die Berechnungen, die zu den Ausgabespalten führen, können unterschiedlich sein.

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

Der Name der Ansicht muss sich von den Namen aller anderen Beziehungen (Tabelle, Index, Ansicht) im selben Schema unterscheiden.

Parameters

`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 ...;` entspricht `CREATE 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 Namensliste der Ansichtsspalte angegeben werden.

name

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

column_name

Eine optionale Liste von Namen, die für die Spalten in der Ansicht verwendet werden sollen. Wenn keine Spaltennamen angegeben werden, werden die diese 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 einer Angabe von `WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)` – 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 denen des Besitzers der Ansicht verglichen werden. Alle Einzelheiten finden Sie in den folgenden Hinweisen.

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

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, werden die `INSERT`- und `UPDATE`-Befehle in der Ansicht überprüft, um sicherzustellen, dass neue Zeilen die Bedingung erfüllen, die die Ansicht definiert (das heißt, die neuen Zeilen werden auf Sichtbarkeit in der Ansicht überprüft). Sind sie nicht sichtbar, dann wird die Aktualisierung abgelehnt. Wenn kein `CHECK OPTION` angegeben ist, dürfen die Befehle `INSERT` und `UPDATE` neue Zeilen für die Ansicht erstellen, die in der Ansicht nicht sichtbar sind.

`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 Basisansichten geprüft. Wenn die `CHECK OPTION` angegeben ist und sowohl `LOCAL` als auch `CASCADED` nicht angegeben sind, wird von `CASCADED` ausgegangen.

> **Note**
>
> Die `CHECK OPTION` darf in `RECURSIVE`-Ansichten nicht verwendet werden. Die `CHECK OPTION` wird ausschließlich für Ansichten unterstützt, die automatisch aktualisierbar sind.

Hinweise

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

Die Namen und Datentypen der Ansichtsspalten sollten sorgfältig überlegt werden. Zum Beispiel ist die Anweisung CREATE VIEW vista AS SELECT 'Hello World'; nicht zu empfehlen, da der Spaltenname standardmäßig auf `?column?;` gesetzt wird. Außerdem wird der Datentyp der Spalte automatisch auf `text` festgelegt, was vielleicht nicht in Ihrem Sinne wäre.

Ein besserer Ansatz wäre, den Spaltennamen und den Datentyp explizit anzugeben, zum Beispiel: `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

Standardmäßig wird der Zugriff auf die zugrunde liegenden Basistabellen, die in der Ansicht referenziert werden, durch die Berechtigungen des Ansichtenbesitzers bestimmt. In einigen Fällen kann auf diese Weise ein sicherer, jedoch eingeschränkter Zugriff auf die zugrunde liegenden Tabellen ermöglicht werden. 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 ausführenden Benutzers bestimmt, und nicht durch den Besitzer der Ansicht. Daher muss ein Benutzer einer Security-Invoker-Ansicht über die entsprechenden Berechtigungen sowohl für die Ansicht als auch für die zugrunde liegenden Basistabellen verfügen.
- Wenn es sich bei einer der zugrunde liegenden Basisbeziehungen um eine Security-Invoker-Ansicht handelt, wird sie so behandelt, als ob direkt von der ursprünglichen Abfrage aus auf sie zugegriffen worden wäre. Daher überprüft eine Security-Invoker-Ansicht ihre zugrunde liegenden

Basistabellen stets anhand der Berechtigungen des aktuellen Benutzers, auch wenn sie von einer Ansicht ohne die `security_invoker`-Eigenschaft aufgerufen 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 ausführenden Benutzers oder des Funktionsbesitzers der Ansicht ausgeführt, je nachdem, ob die Funktionen als `SECURITY INVOKER` oder `SECURITY DEFINER` definiert sind.
- Der Benutzer, der eine View erstellt oder ersetzt, muss über `USAGE`-Berechtigungen für alle Schemas verfügen, die in der Ansichtenabfrage referenziert werden, um die darin enthaltenen Objekte auflösen zu können.
- Wenn `CREATE OR REPLACE VIEW` für eine bestehende Ansicht verwendet wird, werden nur die definierende `SELECT`-Regel, etwaige `WITH ( ... )`-Parameter sowie deren `CHECK OPTION` geändert. Andere Ansichteneigenschaften, einschließlich Besitzrechte, Berechtigungen und non-SELECT-Regeln bleiben unverändert. Um eine Ansicht zu ersetzen, müssen Sie der Ansichtenbesitzer sein (das schließt auch eine Mitgliedschaft in der besitzenden Rolle ein).

Aktualisierbare Ansichten

Einfache Ansichten sind automatisch aktualisierbar: das System erlaugt die Verwendung von `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen auf der Ansicht genau wie bei 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 Ansichtendefinition darf auf der obersten Ebene keine `WITH`-,`DISTINCT`-, `GROUP BY`-, `HAVING`-, `LIMIT`- oder `OFFSET`-Klauseln enthalten.
- Die Ansichtendefinition darf auf oberster Ebene keine Mengenoperationen wie `UNION`, `INTERSECT` oder `EXCEPT` 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 Basisbeziehung handelt. Andernfalls

ist die Spalte schreibgeschützt, und es tritt ein Fehler auf, wenn eine `INSERT`- oder `UPDATE`-Anweisung versucht, ihr einen Wert zuzuweisen.

Eine komplexere Ansicht, die nicht alle Voraussetzungen für automatische Aktualisierbarkeit erfüllt, ist standardmäßig schreibgeschützt – das heißt, das System erlaubt keine INSERT-, UPDATE- oder DELETE-Operationen auf dieser Ansicht.

> **Note**
>
> Der Benutzer, der ein INSERT, UPDATE oder DELETE auf einer Ansicht ausführt, muss die entsprechenden Berechtigungen für diese Operationen auf der Ansicht selbst besitzen Standardmäßig muss der Besitzer der Ansicht über die notwendigen Rechte auf die zugrunde liegenden Basistabellen verfügen. Der Benutzer, der die Ansicht verwendet, benötigt keine direkten Berechtigungen für diese Basistabellen. Wenn die Ansicht jedoch mit security_invoker = true definiert ist, muss der Benutzer, der die Aktualisierung ausführt, und nicht der Ansichtenbesitzer die entsprechenden Rechte auf die zugrunde liegenden Basistabellen besitzen.

Beispiele

Erstellen einer Ansicht mit allen Comedy-Filmen

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

Erstellen einer Ansicht mit `LOCAL CHECK OPTION`

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

Erstellen Sie eine rekursive Ansicht.

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
```

```
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

Kompatibilität

`CREATE OR REPLACE VIEW` ist eine PostgreSQL-Spracherweiterung. Die `WITH ( ... )`-Klausel ist ebenfalls eine Erweiterung, wie auch die Ansichten Sicherheitsbarriere und Sicherheitsaufrufer. 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 [, ... ] )
```

Description

`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 wechseln, müssen Sie berechtigt sein, `SET ROLE` auf die neue Besitzerrolle zu setzen, und diese Rolle muss über `CREATE`-Berechtigungen für das Schema der Ansicht verfügen.

Parameters

`name`

Der Name (optional schemaqualifiziert) einer vorhandenen Ansicht.

`column_name`

Name einer vorhandenen Spalte oder neuer Name für eine vorhandene Spalte.

`IF EXISTS`

Geben keine Fehler aus, wenn die Ansicht nicht vorhanden ist. In diesem Fall wird eine Mitteilung ausgegeben.

`SET/DROP DEFAULT`

Diese Formulare legen den Standardwert für eine Spalte fest oder entfernen ihn. Der Standardwert für eine Ansichtsspalte wird durch einen beliebigen `INSERT`- oder `UPDATE`-Befehl ersetzt, bei dem das Ziel die Ansicht ist.

`new_owner`

Der Benutzername des neuen Ansichtenbesitzers.

`new_name`

Der neue Name der Ansicht.

`new_schema`

Das neue Schema der Ansicht.

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

Legt eine Anzeigeoption fest. 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 Sicherheitsbarriereneigenschaft der Ansicht.
- `security_invoker (boolean)`- Ändert die Eigenschaft Security-Invoker der Ansicht.

`RESET ( view_option_name [, ... ] )`

Setzt eine Ansichtsoption auf ihren Standardwert zurück.

Beispiele

Die Ansicht `foo` wird umbenannt in: `bar`

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

Einen Standardspaltenwert an eine aktualisierbare Ansicht anhängen:

```
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 ]
```

Description

`DROP VIEW` löscht eine vorhandene Ansicht. Sie müssen der Ansichtenbesitzer sein, um diesen Befehl ausführen zu können.

Parameters

IF EXISTS

Geben keine Fehler aus, wenn die Ansicht nicht vorhanden ist. In diesem Fall wird eine Mitteilung ausgegeben.

name

Der Name (optional schemaqualifiziert) der zu entfernenden Ansicht.

CASCADE

Objekte, die von einer Ansicht abhängig sind (z. B. von anderen Ansichten), können automatisch gelöscht werden, einschließlich aller weiteren Objekte, die von diesen abhängigen Objekten abhängig sind.

`RESTRICT`

Verweigert die Löschung einer Ansicht, wenn andere Objekte von ihr abhängig sind. Dies ist die Standardeinstellung.

Beispiele

```
DROP VIEW kinds;
```

Kompatibilität

Dieser Befehl entspricht dem SQL-Standard, mit zwei Ausnahmen: erstens ist nur das Löschen einer einzelnen Ansicht pro Befehl erlaubt, und zweitens wenn die Option `IF EXISTS` eine PostgreSQL-Erweiterung ist, die auch von Aurora DSQL unterstützt wird.

Migration von PostgreSQL zu Aurora DSQL

Aurora DSQL ist so konzipiert, dass es PostgreSQL-kompatibel ist und relationale Kernfunktionen wie ACID-Transaktionen, Sekundärindizes, Joins und Standard-DML-Operationen unterstützt. Die meisten vorhandenen PostgreSQL-Anwendungen können mit minimalen Änderungen zu Aurora DSQL migriert werden.

Dieser Abschnitt enthält praktische Anleitungen für die Migration Ihrer Anwendung zu Aurora DSQL, einschließlich Framework-Kompatibilität, Migrationsmustern und architektonischen Überlegungen.

Framework- und ORM-Kompatibilität

Aurora DSQL verwendet das standardmäßige PostgreSQL-Wire-Protokoll und gewährleistet so die Kompatibilität mit PostgreSQL-Treibern und Frameworks. Die beliebtesten ORMs arbeiten mit Aurora DSQL mit minimalen oder keinen Änderungen. Referenzimplementierungen und verfügbare ORM-Integrationen finden the section called “Aurora DSQL-Adapter” Sie unter.

Allgemeine Migrationsmuster

Bei der Migration von PostgreSQL zu Aurora DSQL funktionieren einige Funktionen anders oder haben eine alternative Syntax. Dieser Abschnitt enthält Anleitungen zu gängigen Migrationsszenarien.

Alternativen zum DDL-Betrieb

Aurora DSQL bietet moderne Alternativen zu herkömmlichen PostgreSQL-DDL-Operationen:

Erstellung von Indizes

Verwenden Sie `CREATE INDEX ASYNC` statt `CREATE INDEX` für die blockierungsfreie Indexerstellung.

Vorteil: Indexerstellung für große Tabellen ohne Ausfallzeiten.

Entfernung von Daten

Verwenden Sie `DELETE FROM table_name` stattTRUNCATE.

Alternative: Verwenden Sie zur vollständigen Neuerstellung der Tabelle `DROP TABLE` gefolgt von`CREATE TABLE`.

Konfiguration des Systems

Aurora DSQL wird vollständig verwaltet, sodass die Konfiguration automatisch auf der Grundlage von Workload-Mustern erfolgt. Verwenden Sie die AWS Managementkonsole oder API, um die Cluster-Einstellungen zu verwalten.

Vorteil: Datenbankoptimierung oder Parameterverwaltung sind nicht erforderlich.

Schema-Entwurfsmuster

Passen Sie diese gängigen PostgreSQL-Muster an, um die Aurora DSQL-Kompatibilität zu gewährleisten:

Referenzielle Integritätsmuster

Aurora DSQL unterstützt Tabellenbeziehungen und `JOIN` Operationen. Um die referenzielle Integrität zu gewährleisten, implementieren Sie die Validierung in Ihrer Anwendungsebene. Dieses Design entspricht modernen Mustern verteilter Datenbanken, bei denen die Validierung auf Anwendungsebene mehr Flexibilität bietet und Leistungsengpässe durch kaskadierende Operationen vermeidet.

Muster: Implementieren Sie referenzielle Integritätsprüfungen in Ihrer Anwendungsebene unter Verwendung einheitlicher Namenskonventionen, Validierungslogik und Transaktionsgrenzen. Viele anspruchsvolle Anwendungen bevorzugen diesen Ansatz, um die Fehlerbehandlung und Leistung besser kontrollieren zu können.

Temporäre Datenverarbeitung

Verwenden Sie CTEs Unterabfragen oder reguläre Tabellen mit Bereinigungslogik anstelle von temporären Tabellen.

Alternative: Erstellen Sie Tabellen mit sitzungsspezifischen Namen und bereinigen Sie sie in Ihrer Anwendung.

Architektonische Unterschiede verstehen

Die verteilte, serverlose Architektur von Aurora DSQL unterscheidet sich bewusst in mehreren Bereichen von herkömmlichem PostgreSQL. Diese Unterschiede ermöglichen die wichtigsten Vorteile von Aurora DSQL in Bezug auf Einfachheit und Skalierbarkeit.

Vereinfachtes Datenbankmodell

Eine einzige Datenbank pro Cluster

Aurora DSQL bietet eine integrierte Datenbank, die `postgres` pro Cluster benannt ist.

Migrationstipp: Wenn Ihre Anwendung mehrere Datenbanken verwendet, erstellen Sie separate Aurora DSQL-Cluster für die logische Trennung oder verwenden Sie Schemas innerhalb eines einzelnen Clusters.

Keine temporären Tabellen

Für die Verarbeitung temporärer Daten SOLLTEN Sie allgemeine Tabellenausdrücke (CTEs) und Unterabfragen verwenden, die flexible Alternativen für komplexe Abfragen bieten.

Alternative: Verwenden Sie CTEs `WITH` Klauseln für temporäre Ergebnismengen oder reguläre Tabellen mit eindeutiger Benennung für sitzungsspezifische Daten.

Automatische Speicherverwaltung

Aurora DSQL macht Tablespaces und manuelles Speichermanagement überflüssig. Der Speicher wird automatisch auf der Grundlage Ihrer Datenmuster skaliert und optimiert.

Vorteil: Sie müssen den Festplattenspeicher nicht überwachen, die Speicherzuweisung planen oder Tablespace-Konfigurationen verwalten.

Moderne Anwendungsmuster

Aurora DSQL fördert moderne Anwendungsentwicklungsmuster, die die Wartbarkeit und Leistung verbessern:

Logik auf Anwendungsebene statt Datenbank-Trigger

Implementieren Sie für triggerähnliche Funktionen eine ereignisgesteuerte Logik in Ihrer Anwendungsebene.

Migrationsstrategie: Verschieben Sie die Triggerlogik in den Anwendungscode, verwenden Sie ereignisgesteuerte Architekturen mit AWS Diensten wie oder implementieren Sie Prüfpfade mithilfe der EventBridge Anwendungsprotokollierung.

SQL-Funktionen für die Datenverarbeitung

Aurora DSQL unterstützt SQL-basierte Funktionen, aber keine prozeduralen Sprachen wie PL/pgSQL.

Alternative: Verwenden Sie SQL-Funktionen für Datentransformationen oder verschieben Sie komplexe Logik auf Ihre Anwendungsebene oder AWS Lambda-Funktionen.

Optimistische Parallelitätssteuerung statt pessimistischer Sperren

Aurora DSQL verwendet Optimistic Concurrency Control (OCC), einen Ansatz ohne Sperren, der sich von herkömmlichen Datenbanksperrmechanismen unterscheidet. Anstatt Sperren zu erwerben, die andere Transaktionen blockieren, ermöglicht Aurora DSQL den Fortgang von Transaktionen ohne Blockierung und erkennt Konflikte beim Festschreiben. Dadurch werden Deadlocks vermieden und langsame Transaktionen daran gehindert, andere Operationen zu blockieren.

Hauptunterschied: Wenn Konflikte auftreten, gibt Aurora DSQL einen Serialisierungsfehler zurück, anstatt Transaktionen auf Sperren warten zu lassen. Dies erfordert, dass Anwendungen eine Wiederholungslogik implementieren, die der Behandlung von Sperrzeitüberschreitungen in herkömmlichen Datenbanken ähnelt. Konflikte werden jedoch sofort gelöst, anstatt zu blockierenden Wartezeiten zu führen.

Entwurfsmuster: Implementieren Sie eine idempotente Transaktionslogik mit Wiederholungsmechanismen. Entwerfen Sie Schemas, um Konflikte zu minimieren, indem Sie zufällige Primärschlüssel verwenden und Updates über Ihren gesamten Schlüsselbereich verteilen. Details hierzu finden Sie unter Parallelitätssteuerung in Aurora DSQL.

Beziehungen und referenzielle Integrität

Aurora DSQL unterstützt Fremdschlüsselbeziehungen zwischen Tabellen, einschließlich `JOIN` Operationen. Um die referenzielle Integrität zu gewährleisten, implementieren Sie die Validierung in Ihrer Anwendungsebene. Die Durchsetzung der referenziellen Integrität kann zwar nützlich sein, aber kaskadierende Operationen (wie das Kaskadieren von Löschungen) können zu unerwarteten Leistungseinbußen führen. Beispielsweise wird das Löschen einer Bestellung mit 1.000 Einzelposten zu einer Transaktion mit 1.001 Zeilen. Viele Kunden vermeiden aus diesem Grund Beschränkungen durch Fremdschlüssel.

Entwurfsmuster: Implementieren Sie referenzielle Integritätsprüfungen in Ihrer Anwendungsebene, verwenden Sie eventuelle Konsistenzmuster oder nutzen Sie AWS Dienste zur Datenvalidierung.

Operative Vereinfachungen

Aurora DSQL macht viele traditionelle Datenbankwartungsaufgaben überflüssig und reduziert so den betrieblichen Aufwand:

Keine manuelle Wartung erforderlich

Aurora DSQL verwaltet automatisch die Speicheroptimierung, die Erfassung von Statistiken und die Leistungsoptimierung. Herkömmliche Wartungsbefehle wie `VACUUM` werden vom System verarbeitet.

Vorteil: Eliminiert die Notwendigkeit von Datenbankwartungsfenstern, Vakuumplanung und Systemparameteroptimierung.

Automatische Partitionierung und Skalierung

Aurora DSQL partitioniert und verteilt Ihre Daten automatisch auf der Grundlage von Zugriffsmustern. Verwenden UUIDs oder von der Anwendung für eine optimale Verteilung generieren IDs .

Migrationstipp: Entfernen Sie die manuelle Partitionierungslogik und lassen Sie Aurora DSQL die Datenverteilung übernehmen. Verwenden UUIDs oder von der Anwendung für eine optimale Verteilung generieren IDs . Wenn für Ihre Anwendung sequenzielle Identifikatoren erforderlich sind, finden Sie weitere Informationen unter. Sequenzen und Identitätsspalten

Agentenmigration mit KI-Tools

KI-Codierungsagenten können Ihre Migration zu Aurora DSQL beschleunigen, indem sie Schemas analysieren, Code transformieren und DDL-Migrationen mit integrierten Sicherheitsprüfungen ausführen.

Kiro für die Migration verwenden

Programmieragenten wie Kiro können Ihnen helfen, Ihren PostgreSQL-Code zu analysieren und zu Aurora DSQL zu migrieren:

- Schemaanalyse: Laden Sie Ihre vorhandenen Schemadateien hoch und bitten Sie Kiro, mögliche Kompatibilitätsprobleme zu identifizieren und Alternativen vorzuschlagen
- Codetransformation: Geben Sie Ihren Anwendungscode an und bitten Sie Kiro, Ihnen zu helfen, die Triggerlogik umzugestalten, Sequenzen durch Transaktionsmuster zu ersetzen oder diese zu ändern UUIDs
- Migrationsplanung: Bitten Sie Kiro, einen step-by-step Migrationsplan zu erstellen, der auf Ihrer spezifischen Anwendungsarchitektur basiert
- DDL-Migrationen: Führen Sie Schemaänderungen mithilfe des Schemas zur Wiederherstellung von Tabellen mit integrierten Sicherheitsprüfungen und Benutzerverifizierung durch

Beispielaufforderungen:

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

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

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

"Drop the legacy_status column from the orders table"

"Change the price column from VARCHAR to DECIMAL in the products table"
```

DDL-Migration mit Neuerstellung von Tabellen

Wenn Sie KI-Agenten mit dem Aurora DSQL MCP-Server verwenden, verwenden bestimmte ALTER TABLE-Operationen ein Tabellenerstellungsmuster, das Ihre Daten sicher migriert. Der Agent kümmert sich um die Komplexität und hält Sie gleichzeitig über jeden Schritt auf dem Laufenden.

Bei den folgenden Vorgängen wird das Muster der Tabellenerstellung verwendet:

Operation	Ansatz
`DROP COLUMN`	Spalte aus neuer Tabelle ausschließen
`ALTER COLUMN TYPE`	Datentyp während der Migration übertragen
`ALTER COLUMN SET/DROP NOT NULL`	Ändern Sie die Einschränkung in der neuen Tabellend efinition
`ALTER COLUMN SET/DROP DEFAULT`	Definieren Sie den Standard in der neuen Tabellendefinition
`ADD/DROP CONSTRAINT`	Integritätsregel in eine neue Tabelle einbeziehen oder entfernen
`MODIFY PRIMARY KEY`	Definieren Sie eine neue PK mit Eindeutigkeitsprüfung
Spalten teilen/zusammenführen	Verwenden Sie SPLIT_PART, SUBSTRING oder CONCAT

Die folgenden ALTER TABLE-Operationen werden direkt ohne Tabellenerstellung unterstützt:

- `ALTER TABLE ... RENAME COLUMN`— Benennt eine Spalte um
- `ALTER TABLE ... RENAME TO`— Benennt eine Tabelle um
- `ALTER TABLE ... ADD COLUMN`— Fügt eine neue Spalte hinzu

Sicherheitsfunktionen: Bei der Ausführung von DDL-Migrationen präsentieren KI-Agenten den Migrationsplan, überprüfen die Datenkompatibilität, bestätigen die Zeilenanzahl und fordern eine ausdrückliche Genehmigung an, bevor zerstörerische Operationen wie DROP TABLE ausgeführt werden.

Batch-Migrationen: Bei Tabellen mit mehr als 3.000 Zeilen stapelt der Agent die Migration automatisch in Schritten von 500 bis 1.000 Zeilen, um die Transaktionslimits einzuhalten.

Aurora DSQL-MCP-Server

Der Aurora DSQL Model Context Protocol (MCP) -Server ermöglicht es KI-Assistenten, sich direkt mit Ihrem Aurora DSQL-Cluster zu verbinden und die Aurora DSQL-Dokumentation zu durchsuchen. Dies ermöglicht der KI:

- Analysieren Sie Ihr bestehendes Schema und schlagen Sie Migrationsänderungen vor
- Führen Sie DDL-Migrationen mit dem Muster zur Wiederherstellung von Tabellen aus
- Testen Sie Abfragen und überprüfen Sie die Kompatibilität während der Migration
- Stellen Sie genaue up-to-date Anleitungen auf der Grundlage der neuesten Aurora DSQL-Dokumentation bereit

Informationen zur Verwendung des Aurora DSQL MCP-Servers mit KI-Assistenten finden Sie in den Setup-Anweisungen für den Aurora DSQL MCP-Server.

Überlegungen zur PostgreSQL-Kompatibilität mit Aurora DSQL

Aurora DSQL unterscheidet sich von der Funktionsunterstützung von selbstverwaltetem PostgreSQL, was die verteilte Architektur, den serverlosen Betrieb und die automatische Skalierung ermöglicht. Die meisten Anwendungen funktionieren innerhalb dieser Unterschiede ohne Änderungen.

Allgemeine Überlegungen finden Sie unter Überlegungen zur Arbeit mit Amazon Aurora DSQL. Kontingente und Limits finden Sie unter Cluster-Kontingente und Datenbanklimits in Amazon Aurora DSQL.

- Aurora DSQL verwendet eine einzige integrierte Datenbank, die `postgres` pro Cluster benannt ist. Erstellen Sie für die logische Trennung separate Aurora DSQL-Cluster oder verwenden Sie Schemas innerhalb eines einzelnen Clusters.
- Die `postgres` Datenbank verwendet die UTF-8-Zeichenkodierung, die eine breite internationale Zeichenunterstützung bietet.
- Die Datenbank verwendet ausschließlich `C`-Sortierung.
- Aurora DSQL verwendet `UTC` als Systemzeitzone. Postgres speichert alle zeitzonenbezogenen Datums- und Uhrzeitangaben intern in UTC. Sie können den `TimeZone` Konfigurationsparameter so einstellen, dass er konvertiert, wie er dem Client angezeigt wird, und er als Standard für die Client-Eingabe dient, die der Server für die interne Konvertierung in UTC verwendet.
- Die Transaktionsisolationsebene ist bei PostgreSQL auf `Repeatable Read` festgelegt.

- Transaktionen haben die folgenden Einschränkungen:
  - DDL- und DML-Operationen erfordern separate Transaktionen
  - Eine Transaktion kann nur 1 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`)
- Das Timeout für Datenbankverbindungen liegt bei 1 Stunde.
- Aurora DSQL verwaltet Berechtigungen über Zuschüsse auf Schemaebene. Admin-Benutzer erstellen Schemas mithilfe von `CREATE SCHEMA` und gewähren Zugriff mithilfe von. `GRANT USAGE ON SCHEMA` Admin-Benutzer verwalten Objekte im öffentlichen Schema, während Benutzer ohne Administratorrechte Objekte in von Benutzern erstellten Schemas erstellen, um klare Eigentumsgrenzen zu erreichen. Weitere Informationen finden Sie unter Autorisieren von Datenbankrollen zur Verwendung von SQL in Ihrer Datenbank.

Benötigen Sie Hilfe bei der Migration?

Wenn Sie auf Funktionen stoßen, die für Ihre Migration wichtig sind, aber derzeit nicht in Aurora DSQL unterstützt werden, finden Sie unter Informationen Feedback zu Amazon Aurora DSQL geben zum Teilen von Feedback mit AWS.

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, sperrenlosen Parallelitätskontrollmechanismus. Dieser gewährleistet vollständige ACID-Konformität durch Snapshot-Isolierung und gewährleistet so Datenkonsistenz und -zuverlässigkeit.

Die sperrenfreie Architektur, die häufig auftretende Engpässe bei der Datenbankleistung beseitigt, ist ein entscheidender Vorteil von Aurora DSQL. 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 Optimistic Concurrency Control (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 die Transaktionen wie folgt:

- Die Transaktion mit der frühesten Festschreibungszeit 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 eine Wiederholungslogik zur Konfliktbehandlung implementieren. Das ideale Entwurfsmuster ist idempotent, sodass Transaktionen wenn möglich immer gleich erneut ausgeführt werden können. Die empfohlene Logik ähnelt dem Abbruch- und Wiederholungsverhalten bei einem PostgreSQL-Sperrzeitlimit oder Deadlock. Allerdings erfordert OCC, dass Ihre Anwendung diese Wiederholungslogik häufiger anwendet.

Richtlinien für die Optimierung der Transaktionsleistung

Um die Leistung zu optimieren, sollte hohe Konkurrenz auf einzelnen Schlüsseln oder kleinen Schlüsselbereichen vermieden werden. Dazu 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 Konflikten bei einzelnen Schlüsseln erhöhen. Dieser Ansatz gewährleistet eine optimale Leistung auch bei steigendem Transaktionsvolumen.

DDL und verteilte Transaktionen in Aurora DSQL

Data Definition Language (DDL) verhält sich in Aurora DSQL anders als in PostgreSQL. Aurora DSQL verfügt über eine verteilte Multi-AZ-Datenbankschicht und Shared-Nothing-Datenbankebene, die auf Mehrmandanten-Rechen- und Speicherflotten aufbaut. Da es keinen einzelnen primären Datenbankknoten oder Leader gibt, wird der Datenbankkatalog verteilt. Somit verwaltet Aurora DSQL DDL-Schemaänderungen als verteilte Transaktionen.

Insbesondere verhält sich DDL in Aurora DSQL wie folgt anders:

Parallelitätsfehler

Aurora DSQL gibt einen Fehler zur Verletzung der Parallelitätskontrolle aus, wenn eine Transaktion ausgeführt wird, während eine andere dieselbe Ressource aktualisiert. Betrachten Sie beispielsweise die folgende Abfolge von Aktionen:

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

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

DDL und DML in derselben Transaktion

Transaktionen in Aurora DSQL können lediglich eine DDL-Anweisung und nicht gleichzeitig DDL- und DML-Anweisungen enthalten. Diese Einschränkung bedeutet, dass Sie nicht innerhalb einer und derselben Transaktion eine Tabelle erstellen und Daten in die 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 sowohl `CREATE`- als auch `INSERT`-Anweisungen enthält.

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

Asynchrone DDL

Im Standard-PostgreSQL sperren DDL-Operationen wie die `CREATE INDEX`-Sperre die betroffene Tabelle, sodass sie für Lese- und Schreibzugriffe 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 über den asynchronen Job Manager in Aurora DSQL finden Sie unter Asynchrone Indizes in Aurora DSQL.

Primärschlüssel in Aurora DSQL

In Aurora DSQL ist ein Primärschlüssel ein Feature, mit dem Tabellendaten physisch organisiert werden. Es ähnelt der `CLUSTER`-Operation in PostgreSQL oder einem geclusterten Index in anderen Datenbanken. Wenn Sie einen Primärschlüssel definieren, erstellt Aurora DSQL einen Index mit allen Spalten in der Tabelle. Die Primärschlüsselstruktur in Aurora DSQL gewährleistet eine effiziente Datenzugriffs- und Verwaltungsstrategie.

Datenstruktur und Speicherung

Wenn Sie einen Primärschlüssel definieren, speichert Aurora DSQL Tabellendaten in der Reihenfolge der Primärschlüssel. Diese indexorganisierte Struktur ermöglicht es, bei einer Primärschlüsselabfrage alle Spaltenwerte direkt abzurufen, anstatt wie bei einem herkömmlichen B-Baum-Index einem Zeiger zur eigentlichen Datenzeile zu folgen. Im Gegensatz zur `CLUSTER`-Operation in PostgreSQL, die Daten nur einmalig reorganisiert, hält Aurora DSQL diese Ordnung automatisch und fortlaufend aufrecht. Dieser Ansatz verbessert die Leistung von Abfragen, die auf den Primärschlüsselzugriff angewiesen sind.

Aurora DSQL verwendet den Primärschlüssel außerdem, um für jede Zeile in Tabellen und Indizes einen clusterweit eindeutigen Schlüssel zu generieren. Dieser eindeutige Schlüssel bildet die Grundlage für das verteilte Datenmanagement: Er ermöglicht die automatische Partitionierung von Daten über mehrere Knoten hinweg, wobei skalierbaren Speicherung und hohe Parallelität unterstützt wird. Dadurch trägt die Primärschlüsselstruktur wesentlich dazu bei, dass Aurora DSQL automatisch skaliert und gleichzeitige Workloads effizient verwaltet.

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. Diesen Schlüssel können Sie später nicht mehr ändern und können auch keinen 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 versteckte synthetische ID zu.

- Für Tabellen mit hohem Schreibaufkommen sollten monoton steigende Ganzzahlen nicht als Primärschlüssel verwendet werden. 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 weist viele eng beieinander liegende oder doppelte Werte auf. Ein aufsteigender Schlüssel kann auch dann verwendet werden, wenn er dicht ist, da die Schreibperformance in diesem Fall weniger kritisch ist.
- Sollte ein vollständiger Tabellenscan Ihre Leistungsanforderungen nicht erfüllen, dann wählen Sie eine effizientere Zugriffsmethode. In den meisten Fällen bedeutet das die Verwendung eines Primärschlüssels, der Ihrem häufigsten Join- und Lookup-Schlüssel in Abfragen entspricht.
- Die maximale kombinierte Größe der 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.

Sequenzen und Identitätsspalten

Sequenzen und Identitätsspalten generieren Ganzzahlwerte und sind nützlich, wenn kompakte oder menschenlesbare Bezeichner benötigt werden. Diese Werte beinhalten das in der Dokumentation beschriebene Zuordnungs- und Caching-Verhalten. `CREATE SEQUENCE`

Topics

- Sequenz-Bearbeitungsfunktionen
- Identitätsspalten
- Arbeiten mit Sequenzen und Identitätsspalten

Sequenz-Bearbeitungsfunktionen

In diesem Abschnitt werden Funktionen für die Bearbeitung von Sequenzobjekten beschrieben, die auch Sequenzgeneratoren oder einfach Sequenzen genannt werden. Sequenzobjekte sind spezielle einzeilige Tabellen, die mit `CREATE SEQUENCE` erstellt wurden. Sequenzobjekte werden häufig verwendet, um eindeutige Bezeichner für Zeilen einer Tabelle zu generieren. Die Sequenzfunktionen bieten einfache, für mehrere Benutzer sichere Methoden zum Abrufen aufeinanderfolgender Sequenzwerte aus Sequenzobjekten.

> **Important**
>
> Bei der Verwendung von Sequenzen sollte der Cache-Wert sorgfältig abgewogen werden. Weitere Informationen finden Sie im Callout „Wichtig“ auf der `CREATE SEQUENCE` Seite. Hinweise zur optimalen Verwendung von Sequenzen, die auf Workload-Mustern basieren, finden Sie unterArbeiten mit Sequenzen und Identitätsspalten.

Funktion	Description
`nextval ( regclass ) # bigint`	Bringt das Sequenzobjekt auf seinen nächsten Wert zurück und gibt diesen Wert zurück. Dies erfolgt atomar: Selbst wenn mehrere Sitzungen `nextval` gleichzei tig ausgeführt werden, erhält jede Sitzung sicher einen eigenen Sequenzwert. Wenn das Sequenzobjekt mit Standardparametern erstellt wurde, geben aufeinand erfolgende `nextval` Aufrufe steigende Werte zurück, die mit 1 beginnen. Andere Verhaltensweisen können durch die Verwendung geeigneter Parameter im `CREATE SEQUENCE` Befehl erreicht werden. Diese Funktion erfordert `USAGE` unsere `UPDATE` Rechte für die Sequenz.
`setval ( regclass, bigint [, boolean ] ) # bigint`	Legt den aktuellen Wert des Sequenzobjekts und optional sein `is_called` Flag fest. Das Formular mit zwei Parametern setzt das `last_value` Feld der Sequenz auf den angegebenen Wert und sein `is_called` Feld auf`true`, was bedeutet, dass das nächste Feld die Sequenz weiterleitet, bevor ein Wert zurückgegeben

Funktion	Description

	`nextval` wird. Der Wert, der gemeldet wird, `currval` wird ebenfalls auf den angegebenen Wert gesetzt. `is_called` Kann in der Form mit drei Parametern entweder auf `true` oder `false` gesetzt werden. `true`hat den gleichen Effekt wie das Formular mit zwei Parameter n. Wenn es auf gesetzt ist`false`, gibt der nächste `nextval` Wert genau den angegebenen Wert zurück, und die Weiterentwicklung der Sequenz beginnt mit dem Folgenden. `nextval` Außerdem wird der von gemeldete Wert hier `currval` nicht geändert. Beispiel:
	`SELECT setval('myseq', 42); -- Next nextval will return 43`
	`SELECT setval('myseq', 42, true); -- Same as above`
	`SELECT setval('myseq', 42, false); -- Next nextval will return 42`
	Das von zurückgegebene Ergebnis `setval` ist nur der Wert des zweiten Arguments. Für diese Funktion sind `UPDATE` Zugriffsrechte für die Sequenz erforderlich.

`currval ( regclass ) # bigint`	Gibt den zuletzt von `nextval` für diese Sequenz in der aktuellen Sitzung erhaltenen Wert zurück. (Es wird ein Fehler gemeldet, wenn `nextval` diese Sequenz in dieser Sitzung noch nie aufgerufen wurde.) Da dies einen sitzungslokalen Wert zurückgibt, gibt es eine vorherseh bare Antwort darauf, ob andere Sitzungen `nextval` aufgrund der aktuellen Sitzung ausgeführt wurden oder nicht. Diese Funktion erfordert unsere `USAGE SELECT` Rechte für die Sequenz.

Funktion	Description
`lastval () # bigint`	Gibt den Wert zurück, der zuletzt von `nextval` in der aktuellen Transaktion zurückgegeben wurde. Diese Funktion ist identisch mit`currval`, außer dass sie nicht den Sequenznamen als Argument verwendet, sondern sich auf die Sequenz bezieht, auf die in der aktuellen Transaktion zuletzt angewendet `nextval` wurde. Es ist ein Fehler, sie aufzurufen`lastval`, wenn sie in der aktuellen Transaktion `nextval` noch nicht aufgerufen wurde. Diese Funktion erfordert `USAGE` unsere `SELECT` Rechte für die zuletzt verwendete Sequenz.

Warning

Der von abgerufene Wert wird `nextval` nicht zur Wiederverwendung zurückgefordert, wenn die aufrufende Transaktion später abgebrochen wird. Das bedeutet, dass Transaktionsabbrüche oder Datenbankabstürze zu Lücken in der Reihenfolge der zugewiesenen Werte führen können. Das kann auch ohne einen Transaktionsabbruch passieren. Zum Beispiel berechnet eine `INSERT ON CONFLICT` withan-Klausel das to-be-inserted Tupel, einschließlich aller erforderlichen `nextval` Aufrufe, bevor ein Konflikt erkannt wird, der dazu führen würde, dass sie stattdessen der `ON CONFLICT` Regel folgt. Daher können die Sequenzobjekte von Aurora DSQL nicht verwendet werden, um „lückenlose“ Sequenzen zu erhalten.
Ebenso `setval` sind Änderungen des Sequenzstatus, die von vorgenommen wurden, für andere Transaktionen sofort sichtbar und werden nicht rückgängig gemacht, wenn die aufrufende Transaktion zurückgesetzt wird.

Die Sequenz, mit der eine Sequenzfunktion arbeiten soll, wird durch ein `regclass` Argument angegeben, das einfach die OID der Sequenz im `pg_class` Systemkatalog ist. Sie müssen die OID jedoch nicht manuell nachschlagen, da der Eingangskonverter des `regclass` Datentyps die Arbeit für Sie erledigt. Einzelheiten finden Sie in der PostgreSQL-Dokumentation zu Object Identifier Types.

Identitätsspalten

> **Important**
>
> Bei der Verwendung von Identitätsspalten sollte der Cache-Wert sorgfältig abgewogen werden. Weitere Informationen finden Sie in der Erläuterung „Wichtig“ auf der CREATE SEQUENCE Seite.
> Hinweise zur optimalen Verwendung von Identitätsspalten auf der Grundlage von Workload-Mustern finden Sie unterArbeiten mit Sequenzen und Identitätsspalten.

Eine Identitätsspalte ist eine spezielle Spalte, die automatisch aus einer impliziten Sequenz generiert wird. Sie kann verwendet werden, um Schlüsselwerte zu generieren. Um eine Identitätsspalte zu erstellen, verwenden Sie die `GENERATED ... AS IDENTITY` Klausel CREATE TABLE beispielsweise in:

```
CREATE TABLE people (
    id bigint GENERATED ALWAYS AS IDENTITY (CACHE 70000),
    ...
);
```

oder alternativ:

```
CREATE TABLE people (
    id bigint GENERATED BY DEFAULT AS IDENTITY (CACHE 70000),
    ...
);
```

Weitere Details finden Sie unter CREATE TABLE.

Wenn ein `INSERT` Befehl für die Tabelle mit der Identitätsspalte ausgeführt wird und kein Wert explizit für die Identitätsspalte angegeben ist, wird ein durch die implizite Sequenz generierter Wert eingefügt. Zum Beispiel mit den obigen Definitionen und unter der Annahme zusätzlicher geeigneter Spalten, schreiben Sie:

```
INSERT INTO people (name, address) VALUES ('A', 'foo');
INSERT INTO people (name, address) VALUES ('B', 'bar');
```

würde Werte für die `id` Spalte ab 1 generieren und zu den folgenden Tabellendaten führen:

```
 id | name | address
----+------+---------
  1 | A    | foo
  2 | B    | bar
```

Alternativ DEFAULT kann das Schlüsselwort anstelle eines Werts angegeben werden, um den sequenzgenerierten Wert explizit anzufordern:

```
INSERT INTO people (id, name, address) VALUES (DEFAULT, 'C', 'baz');
```

In ähnlicher Weise DEFAULT kann das Schlüsselwort in UPDATE Befehlen verwendet werden.

Somit verhält sich eine Identitätsspalte in vielerlei Hinsicht wie eine Spalte mit einem Standardwert.

Die Klauseln ALWAYS und BY DEFAULT die Spaltendefinition legen fest, wie explizit benutzerdefinierte Werte in INSERT Befehlen und behandelt werden. UPDATE Wenn in einem INSERT Befehl ausgewählt ALWAYS ist, wird ein benutzerdefinierter Wert nur akzeptiert, wenn in der INSERT Anweisung Folgendes angegeben ist. OVERRIDING SYSTEM VALUE Wenn ausgewählt BY DEFAULT ist, hat der benutzerdefinierte Wert Vorrang. Daher BY DEFAULT führt die Verwendung zu einem Verhalten, das Standardwerten ähnlicher ist, bei dem der Standardwert durch einen expliziten Wert außer Kraft gesetzt werden kann, während ALWAYS der Schutz vor dem versehentlichen Einfügen eines expliziten Werts etwas besser ist.

Der Datentyp einer Identitätsspalte muss einer der Datentypen sein, die von Sequenzen unterstützt werden. (Siehe CREATE SEQUENCE). Die Eigenschaften der zugehörigen Sequenz können beim Erstellen einer Identitätsspalte angegeben (siheCREATE TABLE) oder danach geändert werden (sieheALTER TABLE).

Eine Identitätsspalte wird automatisch als markiertNOT NULL. Eine Identitätsspalte garantiert jedoch keine Eindeutigkeit. (Eine Sequenz gibt normalerweise eindeutige Werte zurück, aber eine Sequenz könnte zurückgesetzt werden, oder Werte könnten manuell in die Identitätsspalte eingefügt werden, wie zuvor beschrieben.) Die Eindeutigkeit müsste mit einer PRIMARY KEY UNIQUE OR-Einschränkung erzwungen werden.

Arbeiten mit Sequenzen und Identitätsspalten

In diesem Abschnitt erfahren Sie, wie Sie Sequenzen und Identitätsspalten auf der Grundlage von Workload-Mustern am besten verwenden können.

> **Important**
>
> Weitere Informationen zum Zuweisungs- und Caching-Verhalten finden Sie im Abschnitt Wichtig auf der `CREATE SEQUENCE` Seite.

Identifier-Typen auswählen

Amazon Aurora DSQL unterstützt sowohl UUID-basierte Identifikatoren als auch Ganzzahlwerte, die mithilfe von Sequenzen oder Identitätsspalten generiert wurden. Diese Optionen unterscheiden sich darin, wie Werte zugewiesen werden und wie sie unter Last skaliert werden.

UUID-Werte können ohne Koordination generiert werden und eignen sich gut für Workloads, bei denen Identifikatoren häufig oder über viele Sitzungen hinweg erstellt werden. Da Amazon Aurora DSQL für den verteilten Betrieb konzipiert ist, ist die Vermeidung von Koordination oft von Vorteil. Aus diesem Grund UUIDs werden sie als Standard-Identifikationstyp empfohlen, insbesondere für Primärschlüssel in Workloads, bei denen Skalierbarkeit wichtig ist und eine strikte Reihenfolge der Identifikatoren nicht erforderlich ist.

Sequenzen und Identitätsspalten generieren kompakte Ganzzahlwerte, die sich für menschenlesbare Kennungen, Berichte und externe Schnittstellen eignen. Wenn numerische Identifikatoren aus Gründen der Benutzerfreundlichkeit oder Integration bevorzugt werden, sollten Sie die Verwendung einer Sequenz- oder Identitätsspalte in Kombination mit UUID-basierten Identifikatoren in Betracht ziehen. Wenn Ganzzahlsequenz- oder Identitätswerte erforderlich sind, wird die Auswahl einer geeigneten Cachegröße zu einem wichtigen Bestandteil des Workload-Designs. Im folgenden Abschnitt finden Sie Anleitungen zur Auswahl einer Cachegröße.

Auswahl einer Cachegröße

Die Auswahl eines geeigneten Cache-Werts ist ein wichtiger Bestandteil der effektiven Verwendung von Sequenzen und Identitätsspalten. Die Cache-Einstellung bestimmt, wie sich die Identifier-Zuweisung unter Last verhält, was sich sowohl auf den Systemdurchsatz als auch darauf auswirkt, wie genau die Werte die Zuweisungsreihenfolge widerspiegeln.

Eine größere Cachegröße von **`CACHE >= 65536`** ist gut geeignet, wenn:

- Identifikatoren werden mit hoher Frequenz generiert
- Viele Sitzungen werden gleichzeitig eingefügt
- Die Arbeitslast kann Lücken und sichtbare Reihenfolgen tolerieren

Beispielsweise profitieren Workloads zur Erfassung großer Ereignisse (wie IoT oder Telemetrie) sowie betriebliche Identifikatoren wie Auftragsausführung IDs, Referenzen zu Supportfällen oder interne Auftragsnummern in der Regel von größeren Cachegrößen, bei denen Kennungen häufig generiert werden und eine strikte Reihenfolge nicht erforderlich ist.

Eine Cachegröße von 1 ist besser abgestimmt, wenn:

- Die Zuweisungsraten sind relativ niedrig
- Es wird davon ausgegangen, dass sich die Identifikatoren im Laufe der Zeit stärker an die Reihenfolge der Zuteilung halten
- Die Minimierung von Lücken ist wichtiger als maximaler Durchsatz

Workloads wie die Zuweisung von Konto- oder Referenznummern, bei denen Kennungen seltener generiert werden und eine engere Reihenfolge wünschenswert ist, eignen sich besser für eine Cachegröße von 1.

Asynchrone Indizes in Aurora DSQL

Der `CREATE INDEX ASYNC`-Befehl erstellt einen Index für eine oder mehrere Spalten einer vorgegebenen Tabelle. Dieser Befehl ist ein asynchroner DDL-Vorgang, der andere Transaktionen nicht blockiert. Wenn Sie `CREATE INDEX ASYNC` ausführen, gibt Aurora DSQL sofort eine `job_id` zurück.

Sie können den Status dieses asynchronen Auftrags mithilfe der `sys.jobs`-Systemansicht überwachen. Während der Indexerstellungsauftrag ausgeführt wird, 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 Auftrag abgeschlossen ist oder fehlschlägt. Gibt einen Booleschen Wert zurück, der Erfolg oder Misserfolg anzeigt.

`DROP INDEX`

Bricht einen aktiven Index-Erstellungsjob 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 zu zeigen, 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 ]
```

Parameters

`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 soll, wenn bereits ein Index mit demselben Namen vorhanden ist. 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 bereits vorhandene Index haben kann. Wenn Sie diesen Parameter vorgeben, muss der Indexname angegeben werden.

`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 der Ansicht muss sich von den Namen aller anderen Objekte (z. B. Tabelle, Index, Ansicht) im selben 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 `CREATE INDEX ASYNC on table1 (col1, col2)` ausführen, nennt Aurora DSQL den Index automatisch `table1_col1_col2_idx`.

`NULLS FIRST | LAST`

Die Sortierreihenfolge von NULL- und Nicht-NULL-Spalten: `FIRST` gibt an, dass Aurora DSQL NULL-Spalten vor Nicht-NULL-Spalten sortieren soll. `LAST` bedeutet, dass Aurora DSQL NULL-Spalten nach Nicht-NULL-Spalten sortiert.

`INCLUDE`

Eine Liste von Spalten, die im Index als Nicht-Schlüsselspalten enthalten sein sollen. Nicht-Schlüsselspalten können nicht in einer Index-Scan-Suchbedingung verwendet werden. Aurora DSQL ignoriert die Spalte im Hinblick auf die Eindeutigkeit eines Index.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Legt fest, ob Aurora DSQL NULL-Werte in einem eindeutigen Index als eindeutig behandeln soll. Standardmäßig gilt `DISTINCT`, was bedeutet, dass ein eindeutiger Index mehrere NULL-Werte in einer Spalte enthalten darf. `NOT DISTINCT` hingegen gibt an, dass ein Index keine mehrfachen NULL-Werte in einer Spalte zulassen darf.

Nutzungshinweise

Berücksichtigen Sie die folgenden Hinweise:

- Der `CREATE INDEX ASYNC`-Befehl führt keine Sperren ein. Er 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. `sys.jobs` zeigt also 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 `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.

Erstellen eines Index: Beispiel

Im folgenden Beispiel wird gezeigt, 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. Fügen Sie 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 Befehl `CREATE INDEX` gibt eine Auftrags-ID zurück, wie im Folgenden dargestellt.

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

   `job_id` zeigt an, dass Aurora DSQL einen neuen Auftrag zur Indexerstellung eingereicht hat. Sie können das Verfahren `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* verwenden, um andere Arbeiten an der Sitzung zu blockieren, bis der Auftrag abgeschlossen ist oder ein Timeout eintritt.

Abfragen des Indexerstellungsstatus: Beispiel

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

```
SELECT * FROM sys.jobs where job_id = 'wqhu6ewifze5xitg3umt24h5ua';
```

Aurora DSQL gibt eine Antwort zurück, die in etwa wie folgt aussieht:

```
           job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
----------------------------+-----------+---------+-------------+----------+-----------
+------------------+------------------------+------------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
```

Die Statusspalte kann einen der folgenden Werte enthalten:

Status	Description
`submitted`	Die Aufgabe wurde eingereicht, aber Aurora DSQL hat noch nicht mit der Verarbeitung begonnen.
`processing`	Aurora DSQL verarbeitet die Aufgabe.
`failed`	Die Aufgabe ist fehlgeschlagen. Weitere Informationen finden Sie in der Detailspalte. Wenn Aurora DSQL den Index nicht erstellen konnte, wird Aurora DSQL die Indexdefinition nicht automatisch entfernen. Sie müssen den Index manuell mit dem Befehl `DROP INDEX` entfernen.
`completed`	Aurora DSQL hat die Aufgabe erfolgreich abgeschlossen.

Sie können den Indexstatus auch über die Katalogtabellen `pg_index` und `pg_class` abfragen. Insbesondere die Attribute `indisvalid` und `indisimmediate` geben Aufschluss über den aktuellen Indexstatus. Wenn Aurora DSQL Ihren Index erstellt, hat dieser zunächst den Status `INVALID`. Das `indisvalid`-Flag für den Index gibt `FALSE` oder `f` zurück, was darauf hinweist, dass der Index ungültig ist. Wenn das Flag `TRUE` oder `t` zurückgibt, 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 beim Erstellen eines eindeutigen Indexes

Wenn Ihr asynchroner Job zum Erstellen eines eindeutigen Indexes den Status FAILED mit dem Detail `Found duplicate key while validating index for UCVs` anzeigt, bedeutet dies, dass der eindeutige Index aufgrund von Verstößen gegen die Eindeutigkeitsbedingung nicht erstellt werden konnte.

So beheben Sie Fehler beim Erstellen eines eindeutigen Indexes

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

So erkennen Sie Eindeutigkeitsverstöße in Primärtabellen

Die folgende SQL-Abfrage hilft Ihnen bei der Identifizierung von doppelten Werten in einer bestimmten Spalte Ihrer Tabelle. Dies ist besonders nützlich, wenn Sie Eindeutigkeit für eine Spalte erzwingen möchten, die derzeit nicht als Primärschlüssel definiert ist oder keine eindeutige Einschränkung besitzt – etwa bei E-Mail-Adressen in einer Benutzertabelle.

In den folgenden Beispielen wird demonstriert, wie Sie eine Beispielbenutzertabelle erstellen, sie mit Testdaten füllen, die bekannte Duplikate enthalten, und dann die Erkennungsabfrage ausführen.

Tabellenschema definieren

```
-- 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 aus

```
-- 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;
```

Zeigen Sie alle Datensätze mit doppelten E-Mail-Adressen an

```
 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 sowie über nützliche Abfragen zum Abrufen von Informationen über das System, z. B. die Version.

Systemtabellen

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

Wichtige PostgreSQL-Katalogtabellen und -ansichten

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

Name	Description
`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 zu Benutzern
`pg_roles`	Informationen zu Benutzern und Gruppen
`pg_indexes`	Listet alle Indizes auf
`pg_constraint`	Listet Einschränkungen für Tabellen auf

Unterstützte und nicht unterstützte Katalogtabellen

In der folgenden Tabelle sehen Sie, welche Tabellen in Aurora DSQL unterstützt werden und welche nicht.

Name	Gilt für Aurora DSQL
pg_aggregate	Nein
pg_am	Ja
pg_amop	Nein
pg_amproc	Nein
pg_attrdef	Ja
pg_attribute	Ja
pg_authid	Nein (pg_roles verwenden)
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

Name	Gilt für Aurora DSQL
pg_extension	Nein
pg_foreign_data_wrapper	Nein
pg_foreign_server	Nein
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

Name	Gilt für Aurora DSQL
pg_publication_rel	Nein
pg_range	Ja
pg_replication_origin	Nein
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

Name	Gilt für Aurora DSQL
pg_ts_parser	Ja
pg_ts_template	Ja
pg_type	Ja
pg_user_mapping	Nein

Unterstützte und nicht unterstützte Systemansichten

In der folgenden Tabelle sehen Sie, welche Ansichten in Aurora DSQL unterstützt werden 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

Name	Gilt für Aurora DSQL
pg_policies	Nein
pg_prepared_statements	Nein
pg_prepared_xacts	Nein
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

Name	Gilt für Aurora DSQL
pg_user_mappings	Nein
pg_views	Ja
pg_stat_activity	Nein
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

Name	Gilt für Aurora DSQL
pg_statio_all_indexes	Nein
pg_statio_all_sequences	Nein
pg_stat_slru	Nein
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

Name	Gilt für Aurora DSQL
`pg_statio_sys_tables`	Nein
`pg_statio_user_indexes`	Nein

Die sys.jobs-Ansicht

`sys.jobs` bietet Statusinformationen über asynchrone Aufträge. Nachdem Sie beispielsweise einen asynchronen Index erstellt haben, gibt Aurora DSQL `job_uuid` zurück. Sie können diese `job_uuid` im Zusammenhang mit `sys.jobs` verwenden, um den Auftragsstatus zu suchen.

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

Aurora DSQL gibt eine Antwort zurück, die in etwa wie folgt aussieht:

```
           job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
----------------------------+-----------+---------+-------------+----------+-----------
+------------------+------------------------+------------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
 kkngzf33dndl3daacxehpx5eba | completed |         | ANALYZE     |     1259 |     26419
 | public.nt         | 2025-09-25 21:57:05+00 | 2025-09-25 21:57:27+00
 fyopxjb6ovdn7po6lrkj63cyea | completed |         | DROP        |     1259 |     26422
 |                   | 2025-09-25 22:05:57+00 | 2025-09-25 22:06:03+00
```

In der folgenden Tabelle werden die Spalten in der `sys.jobs` Ansicht beschrieben.

sys.jobs: Spalten anzeigen

Spalte	Typ	Description
`job_id`	`text`	Eine Base-32-UUID, die den Job darstellt.
`status`	`text`	Der aktuelle Status des Jobs. Mögliche Werte sind `submitted` , `processing` , `completed` und `failed`. Weitere Informationen finden Sie unter sys.jobs-Statuswerte.

Spalte	Typ	Description
details	text	Alle relevanten Details zum Job. Schlägt der Job fehl, wird ein detaillierter Grund angegeben.
job_type	text	Der Typ des asynchronen Jobs. Mögliche Werte sind: INDEX_BUILD — eine asynchrone Indexerstellung. ANALYZE— ein vom System übermittelter Job zur automatischen Analyse. DROP— entfernt physische Daten nach einer DROP TABLE OR-Operation. DROP INDEX
class_id	oid	Die OID der Katalogtabelle, die das Objekt enthält.
object_id	oid	Die OID des Objekts.
object_name	text	Der vollständig qualifizierte Name des Objekts. DROPJobs können nicht auf bereits gelöschte Objekte verweisen. Wenn ein Objekt, auf das verwiesen wird, bereits gelöscht wurde, object_name kann es NULL sein.
start_time	timestamp with time zone	Der Zeitstempel, zu dem der Job eingereicht wurde.
update_time	timestamp with time zone	Der Zeitstempel, zu dem die Auftragszeile zuletzt aktualisiert wurde.

sys.jobs-Statuswerte

Status	Description
submitted	Die Aufgabe wurde eingereicht, aber Aurora DSQL hat noch nicht mit der Verarbeitung begonnen.
processing	Aurora DSQL verarbeitet die Aufgabe.

Status	Description
`failed`	Die Aufgabe ist fehlgeschlagen. Weitere Informationen finden Sie in der `details` Spalte.
`completed`	Aurora DSQL hat die Aufgabe erfolgreich abgeschlossen.

Die sys.iam_pg_role_mappings-Ansicht

Die Ansicht `sys.iam_pg_role_mappings` enthält Informationen zu den Berechtigungen, die IAM-Benutzern gewährt wurden. Wenn beispielsweise `DQSLDBConnect` eine IAM-Rolle ist, die Aurora DSQL Zugriff auf Nicht-Administratoren gewährt und einem Benutzer namens `testuser` wird the Rolle `DQSLDBConnect` mit den entsprechenden Berechtigungen zugewiesen, können Sie die Systemansicht `sys.iam_pg_role_mappings` abfragen, um zu sehen, welche Benutzer welche Berechtigungen erhalten haben.

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

Nützliche Abfragen von Systemmetadaten

Verwenden Sie diese Abfragen, um Tabellenstatistiken und Systemmetadaten abzurufen, ohne teure Operationen wie vollständige Tabellenscans durchführen zu müssen.

Ermittelt die geschätzte Zeilenanzahl für eine Tabelle

Verwenden Sie die folgende Abfrage, um die ungefähre Anzahl der Zeilen in einer Tabelle zu ermitteln, ohne einen vollständigen Tabellenscan durchzuführen:

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

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

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

Dieser Ansatz ist effizienter als `SELECT COUNT(*)` bei großen Tabellen in Aurora DSQL.

Holen Sie sich die aktuelle Aurora DSQL-Hauptversion

Verwenden Sie die folgende Abfrage, um die aktuelle Hauptversion des Aurora DSQL-Clusters abzurufen:

```
SELECT * FROM sys.dsql_major_version();
```

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

```
 dsql_major_version
--------------------
                  1
```

Dies gibt die Hauptversion zurück, auf der sich die SQL-Verbindung in Aurora DSQL befindet.

Aktuelle PostgreSQL-Version abrufen

Verwenden Sie die folgende Abfrage, um die aktuelle PostgreSQL-Version des Aurora DSQL-Clusters abzurufen:

```
SHOW server_version;
```

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

```
 server_version
----------------
 16.13
```

Dies gibt die PostgreSQL-Version zurück, in der sich die SQL-Verbindung in Aurora DSQL befindet.

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 ab. 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. `ANALYZE` unterliegt nicht dem Timeout-Limit für Datenbanktransaktionen.

Um den manuellen Aufwand zu verringern und Statistiken kontinuierlich aktuell zu halten, führt Aurora DSQL `ANALYZE` automatisch als Hintergrundprozess aus. Dieser Hintergrundprozess 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.

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

Automatisch ausgelöste ANALYZE-Hintergrundprozesse sammeln die gleichen Statistiktypen wie ein manuelles ANALYZE und wenden sie standardmäßig auf Benutzertabellen an. System- und Katalogtabellen sind von diesem automatisierten Prozess ausgenommen.

Mit Aurora DSQL EXPLAIN-Plänen arbeiten

Aurora DSQL verwendet eine ähnliche EXPLAIN-Planstruktur wie PostgreSQL, jedoch mit wichtigen Ergänzungen, die die verteilte Architektur und das Ausführungsmodell widerspiegeln.

In dieser Dokumentation geben wir einen Überblick über die Aurora DSQL EXPLAIN-Pläne und heben die Ähnlichkeiten und Unterschiede im Vergleich zu PostgreSQL hervor. Wir behandeln die verschiedenen Arten von Scanvorgängen, die in Aurora DSQL verfügbar sind, und helfen Ihnen, die Kosten für die Ausführung Ihrer Abfragen zu verstehen.

PostgreSQL VS Aurora DSQL EXPLAIN-Pläne

Aurora DSQL basiert auf der PostgreSQL-Datenbank und teilt die meisten Planstrukturen mit PostgreSQL, weist jedoch wichtige architektonische Unterschiede auf, die sich auf die Ausführung und Optimierung von Abfragen auswirken:

Feature	PostgreSQL	Aurora DSQL
Datenspeicherung	Heap-Speicher	Kein Heap, alle Zeilen sind durch einen eindeutigen Bezeichner indexiert
Primärschlüssel	Der Primärschlüsselindex ist von den Tabellendaten getrennt	Der Primärschlüsselindex ist die Tabelle mit allen zusätzlichen Spalten als INCLUDE-Spalten
Sekundäre Indexe	Standard-Sekundärindizes	Funktioniert genauso wie PostgreSQL, mit der Möglichkeit, Nichtschlüsselspalten einzubezi ehen
Filterfunktionen	Indexbedingung, Heap-Filter	Indexbedingung, Speicherfilter, Abfrageprozessorfilter
Scan-Typen	Sequentieller Scan, Indexscan, Nur Indexscan	Vollständiger Scan, Nur Indexscan , Indexscan
Ausführung der Abfrage	Lokal in der Datenbank	Verteilt (Rechenleistung und Speicher sind getrennt)

Aurora DSQL speichert Tabellendaten direkt in der Reihenfolge der Primärschlüssel und nicht in einem separaten Heap. Jede Zeile wird durch einen eindeutigen Schlüssel identifiziert, in der Regel den Primärschlüssel, wodurch die Datenbank Suchvorgänge effizienter optimieren kann. Der architektonische Unterschied erklärt, warum Aurora DSQL in Fällen, in denen PostgreSQL möglicherweise einen sequentiellen Scan wählt, häufig Index-Only-Scans verwendet.

Ein weiterer wichtiger Unterschied besteht darin, dass Aurora DSQL die Rechenleistung vom Speicher trennt, sodass Filter früher im Ausführungspfad angewendet werden können, um Datenbewegungen zu reduzieren und die Leistung zu verbessern.

Weitere Informationen zur Verwendung von EXPLAIN-Plänen mit PostgreSQL finden Sie in der PostgreSQL EXPLAIN-Dokumentation.

Schlüsselelemente der Aurora DSQL EXPLAIN-Pläne

Aurora DSQL EXPLAIN-Pläne bieten detaillierte Informationen darüber, wie Abfragen ausgeführt werden, einschließlich der Orte, an denen gefiltert wird und welche Spalten aus dem Speicher abgerufen werden. Wenn Sie diese Ausgabe verstehen, können Sie die Abfrageleistung optimieren.

Index Cond

Bedingungen, die für die Navigation im Index verwendet werden. Effizienteste Filterung, die die Anzahl der gescannten Daten reduziert. In Aurora DSQL können Indexbedingungen auf mehreren Ebenen des Ausführungsplans angewendet werden.

Projektionen

Aus dem Speicher abgerufene Spalten. Weniger Prognosen bedeuten bessere Leistung.

Speicherfilter

Auf Speicherebene angewandte Bedingungen. Effizienter als Filter für Abfrageprozessoren.

Prozessorfilter abfragen

Auf der Ebene des Abfrageprozessors angewandte Bedingungen. Erfordert die Übertragung aller Daten vor dem Filtern, was zu einem höheren Datenverschiebungs- und Verarbeitungsaufwand führt.

Filter in Aurora DSQL

Aurora DSQL trennt Rechenleistung vom Speicher, was bedeutet, dass der Punkt, an dem Filter während der Abfrageausführung angewendet werden, erhebliche Auswirkungen auf die Leistung hat. Filter, die vor der Übertragung großer Datenmengen angewendet werden, reduzieren die Latenz und verbessern die Effizienz. Je früher ein Filter angewendet wird, desto weniger Daten müssen verarbeitet, verschoben und gescannt werden, was zu schnelleren Abfragen führt.

Aurora DSQL kann Filter in mehreren Phasen des Abfragepfads anwenden. Das Verständnis dieser Phasen ist entscheidend für die Interpretation von Abfrageplänen und die Optimierung der Leistung.

Level	Typ des Filters	Description
1	Indexbedingung	Wird beim Scannen des Index angewendet. Beschränkt, wie viele Daten aus dem Speicher

Level	Typ des Filters	Description
		gelesen werden, und reduziert die Anzahl der an die Rechenebene gesendeten Daten.
2	Speicherfilter	Wird angewendet, nachdem Daten aus dem Speicher gelesen wurden, aber bevor sie an den Computer gesendet werden. Ein Beispiel hierfür ist ein Filter in einer Include-Spalte eines Indexes. Reduziert die Datenübertragung, aber nicht die Menge, die gelesen wird.
3	Prozessorfilter abfragen	Wird angewendet, nachdem Daten die Rechensch icht erreicht haben. Alle Daten müssen zuerst übertragen werden, was die Latenz und die Kosten erhöht. Derzeit kann Aurora DSQL nicht alle Filter- und Projektionsvorgänge im Speicher durchführen, sodass einige Abfragen möglicherweise gezwungen sind, auf diese Art der Filterung zurückzugreifen.

Aurora DSQL EXPLAIN-Pläne lesen

Zu verstehen, wie man EXPLAIN-Pläne liest, ist der Schlüssel zur Optimierung der Abfrageleistung. In diesem Abschnitt gehen wir reale Beispiele für Aurora DSQL-Abfragepläne durch, zeigen, wie sich verschiedene Scantypen verhalten, erklären, wo Filter angewendet werden, und zeigen Optimierungsmöglichkeiten auf.

In diesen Beispielen verwendete Beispieltabellen

Die folgenden Beispiele verweisen auf zwei Tabellen: `transaction` und`account`.

Die `transaction` Tabelle hat keinen Primärschlüssel, weshalb Aurora DSQL bei der Abfrage vollständige Tabellenscans durchführt.

Für die `account` Tabelle ist ein Index aktiviert. `customer_id` Dieser Index beinhaltet `balance` und `status` als Deckspalten, sodass bestimmte Abfragen direkt vom Index aus beantwortet werden können, ohne aus der Basistabelle lesen zu müssen. Da der Index jedoch keine

Daten enthält`created_at`, benötigen Abfragen, die auf diese Spalte verweisen, zusätzlichen Tabellenzugriff.

```
CREATE TABLE transaction (
    account_id uuid,
    transaction_date timestamp,
    description text
);

CREATE TABLE account (
    customer_id uuid,
    balance numeric,
    status varchar,
    created_at timestamp
);

CREATE INDEX ASYNC idx1 ON account (customer_id) INCLUDE (balance, status);
```

Beispiel für einen vollständigen Scan

Aurora DSQL verfügt sowohl über sequentielle Scans, die funktionell identisch mit PostgreSQL sind, als auch über vollständige Scans. Der einzige Unterschied zwischen diesen beiden besteht darin, dass bei vollständigen Scans zusätzliche Speicherfilterung verwendet werden kann. Aus diesem Grund wird diese Option fast immer über „Sequentielle Scans“ ausgewählt. Aufgrund der Ähnlichkeit werden wir nur Beispiele für die interessanteren vollständigen Scans behandeln.

Vollständige Scans werden hauptsächlich für Tabellen ohne Primärschlüssel verwendet. Da Aurora DSQL-Primärschlüssel standardmäßig Indizes vollständig abdecken, wird Aurora DSQL in vielen Situationen, in denen PostgreSQL einen sequentiellen Scan verwenden würde, höchstwahrscheinlich Index-Only-Scans für den Primärschlüssel verwenden. Wie bei den meisten anderen Datenbanken wird eine Tabelle ohne Indizes schlecht skaliert.

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

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------
 Full Scan (btree-table) on transaction  (cost=125100.05..177933.38 rows=33333
 width=16)
   Filter: (description ~~ '%external%'::text)
   -> Storage Scan on transaction (cost=12510.05..17793.38 rows=66666 width=16)
```

```
        Projections: account_id, description
        Filters: (transaction_date > '2025-01-01 00:00:00'::timestamp without time
 zone)
        -> B-Tree Scan on transaction (cost=12510.05..17793.38 rows=100000 width=30)
```

Dieser Plan zeigt zwei Filter, die in unterschiedlichen Phasen angewendet werden. Die `transaction_date > '2025-01-01'` Bedingung wird auf die Speicherebene angewendet, wodurch die Menge der zurückgegebenen Daten reduziert wird. Die `description LIKE '%external%'` Bedingung wird später im Abfrageprozessor angewendet, nachdem Daten übertragen wurden, wodurch sie weniger effizient ist. Das Verschieben selektiverer Filter in die Speicher- oder Indexebene verbessert im Allgemeinen die Leistung.

Beispiel „Nur Index scannen“

Nur Index-Scans sind die optimalsten Scantypen in Aurora DSQL, da sie zu den wenigsten Roundtrips zur Speicherebene führen und am meisten filtern können. Aber nur weil Sie Index Only Scan sehen, heißt das nicht, dass Sie den besten Plan haben. Aufgrund der unterschiedlichen Filterstufen, die auftreten können, ist es wichtig, trotzdem darauf zu achten, an welchen Stellen gefiltert werden kann.

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

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

In diesem Plan wird die Indexbedingung `customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'` () zuerst während des Indexscans ausgewertet. Dies ist die effizienteste Phase, da sie begrenzt, wie viele Daten aus dem Speicher gelesen werden. Der Speicherfilter,`status = 'pending'`, wird nach dem Lesen der Daten, aber bevor sie an die

Rechenebene gesendet werden, angewendet, wodurch die übertragene Datenmenge reduziert wird. Schließlich wird der Filter für den Abfrageprozessor`balance > 100`,, zuletzt ausgeführt, nachdem die Daten verschoben wurden, was ihn am wenigsten effizient macht. Von diesen bietet die Indexbedingung die beste Leistung, da sie direkt steuert, wie viele Daten gescannt werden.

Beispiel für einen Index-Scan

Index-Scans ähneln Nur-Index-Scans, mit dem Unterschied, dass sie zusätzlich die Basistabelle aufrufen müssen. Da Aurora DSQL Speicherfilter angeben kann, ist dies sowohl beim Indexaufruf als auch beim Suchaufruf möglich.

Um dies zu verdeutlichen, präsentiert Aurora DSQL den Plan als zwei Knoten. Auf diese Weise können Sie deutlich erkennen, wie hilfreich das Hinzufügen einer Include-Spalte in Bezug auf die aus dem Speicher zurückgegebenen Zeilen ist.

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

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

Dieser Plan zeigt, wie die Filterung über mehrere Stufen hinweg erfolgt:

- Die Indexbedingung für `customer_id` filtert Daten frühzeitig.
- Durch den eingeschalteten Speicherfilter werden die Ergebnisse `status` weiter eingegrenzt, bevor sie an die Datenverarbeitung gesendet werden.

- Der aktivierte Filter für den Abfrageprozessor `balance` wird später, nach der Übertragung, angewendet.
- Der aktivierte Suchfilter `created_at` wird beim Abrufen zusätzlicher Spalten aus der Basistabelle ausgewertet.

Durch das Hinzufügen häufig verwendeter Spalten als `INCLUDE` Felder kann diese Suche häufig vermieden und die Leistung verbessert werden.

Bewährte Methoden

- Richten Sie die Filter an den indizierten Spalten aus, um die Filterung früher voranzutreiben.
- Verwenden Sie INCLUDE-Spalten, um reine Index-Scans zu ermöglichen und Suchvorgänge zu vermeiden.
- Überprüfen Sie die Zeilenschätzungen bei der Untersuchung von Leistungsproblemen. Aurora DSQL verwaltet Statistiken automatisch, indem es `ANALYZE` im Hintergrund auf der Grundlage der Datenänderungsraten ausgeführt wird. Wenn Schätzungen ungenau erscheinen, können Sie sie `ANALYZE` manuell ausführen, um die Statistiken sofort zu aktualisieren.
- Vermeiden Sie nicht indizierte Abfragen in großen Tabellen, um teure vollständige Scans zu vermeiden.

Verständnis DPUs in EXPLAIN ANALYZE

Aurora DSQL bietet Informationen zur Distributed Processing Unit (DPU) auf Anweisungsebene in der `EXPLAIN ANALYZE VERBOSE` Planausgabe, sodass Sie während der Entwicklung einen besseren Einblick in die Abfragekosten erhalten. In diesem Abschnitt wird erklärt, was das DPUs sind und wie sie in der Ausgabe zu interpretieren sind. `EXPLAIN ANALYZE VERBOSE`

Was ist eine DPU?

Eine Distributed Processing Unit (DPU) ist das normalisierte Maß für die von Aurora DSQL geleistete Arbeit. Sie besteht aus:

- computedPU — Zeit, die für die Ausführung von SQL-Abfragen aufgewendet wurde
- ReadDPU — Ressourcen, die zum Lesen von Daten aus dem Speicher verwendet werden
- WriteDPU — Ressourcen, die zum Schreiben von Daten in den Speicher verwendet werden

- MultiRegionWriteDPU — Ressourcen, die zur Replikation von Schreibvorgängen auf Peer-Cluster in Konfigurationen mit mehreren Regionen verwendet werden.

Verwendung von DPU in EXPLAIN ANALYZE VERBOSE

Aurora DSQL erweitert sich `EXPLAIN ANALYZE VERBOSE` um eine Schätzung der DPU-Nutzung auf Anweisungsebene bis zum Ende der Ausgabe. Dies bietet einen sofortigen Einblick in die Abfragekosten und hilft Ihnen, Workload-Kostentreiber zu identifizieren, die Abfrageleistung zu optimieren und die Ressourcennutzung besser vorherzusagen.

Die folgenden Beispiele zeigen, wie die in der EXPLAIN ANALYZE VERBOSE-Ausgabe enthaltenen DPU-Schätzungen auf Anweisungsebene interpretiert werden können.

Beispiel 1: SELECT-Abfrage

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

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

In diesem Beispiel führt die SELECT-Anweisung einen reinen Index-Scan durch, sodass der Großteil der Kosten auf Read DPU (0,04312) entfällt, was die aus dem Speicher abgerufenen Daten darstellt, und Compute-DPU (0,01607), was die Rechenressourcen widerspiegelt, die für die Verarbeitung und

Rückgabe der Ergebnisse verwendet werden. Es gibt keine Write-DPU, da die Abfrage keine Daten ändert. Die gesamte DPU (0,05919) ist die Summe aus Compute+Read+Write.

Beispiel 2: INSERT-Abfrage

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

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

Diese Anweisung führt hauptsächlich Schreibvorgänge durch, sodass der Großteil der Kosten mit Write DPU verbunden ist. Die Rechen-DPU (0,01550) steht für die Arbeit, die für die Verarbeitung und das Einfügen der Werte aufgewendet wurde. Der Read DPU (0,00307) spiegelt kleinere Systemlesevorgänge wider (für Katalogsuchen oder Indexprüfungen).

Beachten Sie die Transaktions-Mindestwerte, die neben Lesen und Schreiben angezeigt werden. DPUs Diese geben die Basiskosten pro Transaktion an, die nur anfallen, wenn der Vorgang Lese- oder Schreibvorgänge umfasst. Sie bedeuten nicht, dass für jede Transaktion automatisch eine Gebühr von 0,00375 DPU beim Lesen oder 0,05 beim Schreiben anfällt. Stattdessen werden diese Mindestwerte bei der Kostenaggregation auf Transaktionsebene und nur dann angewendet, wenn Lese- oder Schreibvorgänge innerhalb dieser Transaktion erfolgen. Aufgrund dieses unterschiedlichen Umfangs stimmen Schätzungen auf Kontoauszugsebene `EXPLAIN ANALYZE VERBOSE` möglicherweise nicht exakt mit den in den Daten angegebenen Kennzahlen auf Transaktionsebene oder den Abrechnungsdaten überein. CloudWatch

Verwendung von DPU-Informationen zur Optimierung

DPU-Schätzungen pro Anweisung bieten Ihnen eine leistungsstarke Möglichkeit, Abfragen über die reine Ausführungszeit hinaus zu optimieren. Häufige Anwendungsfälle umfassen:

- Kostenbewusstsein: Verstehen Sie, wie teuer eine Abfrage im Vergleich zu anderen ist.
- Schemaoptimierung: Vergleichen Sie die Auswirkungen von Indizes oder Schemaänderungen auf Leistung und Ressourceneffizienz.
- Budgetplanung: Schätzen Sie die Workload-Kosten auf der Grundlage der beobachteten DPU-Auslastung.
- Abfragevergleich: Bewerten Sie alternative Abfrageansätze anhand ihres relativen DPU-Verbrauchs.

DPU-Informationen interpretieren

Beachten Sie die folgenden bewährten Methoden bei der Verwendung von DPU-Daten von: `EXPLAIN ANALYZE VERBOSE`

- Direkter Einsatz: Verwenden Sie die gemeldete DPU als Möglichkeit, die relativen Kosten einer Abfrage zu verstehen, und nicht als exakte Übereinstimmung mit CloudWatch Kennzahlen oder Abrechnungsdaten. Abweichungen sind zu erwarten, da die Kosten auf `EXPLAIN ANALYZE VERBOSE` Kontoauszugsebene gemeldet werden, während die Aktivitäten auf Transaktionsebene aggregiert werden. CloudWatch CloudWatch umfasst auch Hintergrundoperationen (wie ANALYZE oder Verdichtungen) und Transaktionsaufwand (/), der bewusst ausgeschlossen wurde. `BEGIN COMMIT` `EXPLAIN ANALYZE VERBOSE`
- Die DPU-Variabilität zwischen den Ausführungen ist in verteilten Systemen normal und weist nicht auf Fehler hin. Faktoren wie Zwischenspeicherung, Änderungen des Ausführungsplans, Parallelität oder Verschiebungen in der Datenverteilung können alle dazu führen, dass dieselbe Abfrage von einem Lauf zum nächsten unterschiedliche Ressourcen verbraucht.
- Kleine Operationen stapeln: Wenn Ihr Workload viele kleine Anweisungen ausgibt, sollten Sie erwägen, diese in größeren Vorgängen zu bündeln (nicht mehr als 10 MB). Dies reduziert den Rundungsaufwand und führt zu aussagekräftigeren Kostenschätzungen.
- Wird für die Feinabstimmung verwendet, nicht für die Fakturierung: Die eingehenden `EXPLAIN ANALYZE VERBOSE` DPU-Daten dienen dem Kostenbewusstsein, der Abfrageoptimierung und der Optimierung. Es handelt sich nicht um eine Abrechnungskennzahl. Verlassen Sie sich immer

auf CloudWatch Kennzahlen oder monatliche Abrechnungsberichte, um verlässliche Kosten- und Nutzungsdaten zu erhalten.

Verwalten von Aurora-DB-Clustern

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

Themen

- Konfigurieren von Clustern mit einer einzelnen Region
- Konfigurieren von Clustern mit mehreren Regionen
- Konfigurieren von Aurora DSQL-Clustern mit AWS CloudFormation
- Aurora DSQL-Cluster-Lebenszyklus

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

Konfigurieren von Clustern mit einer einzelnen Region

Konfigurieren und verwalten Sie Cluster für eine AWS-Region entweder über die AWS CLI oder Ihre bevorzugte Programmiersprache , z. B. Python, C++, JavaScript, Java, Rust, Ruby, .NET und Golang. Die AWS CLI bietet schnellen Zugriff über Shell-Befehle, während die AWS-Software Development Kits (SDKs) eine programmatische Steuerung mit nativer Sprachunterstützung ermöglichen.

Themen

- AWS-SDKs verwenden
- Verwenden von AWS-CLI

AWS-SDKs verwenden

Die AWS-SDKs bieten programmatischen Zugriff auf Aurora DSQL in Ihrer bevorzugten Programmiersprache. In den folgenden Abschnitten sehen Sie, wie allgemeine Clusteroperationen mit verschiedenen Programmiersprachen ausgeführt werden.

Cluster erstellen

Die folgenden Beispiele veranschaulichen, wie Sie mithilfe verschiedener Programmiersprachen einen Cluster mit einer Region erstellen.

Python

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

```
import boto3


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

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

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


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


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

C++

Mit folgenden Beispiel können Sie einen Cluster mit einer AWS-Region.

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

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

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

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

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

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

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

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

    return cluster;
}

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

            auto cluster = CreateCluster(region);

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

JavaScript

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

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

async function createCluster(region) {

    const client = new DSQLClient({ region });

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

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

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

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

    await createCluster(region);
}

main();
```

Java

Verwenden Sie das folgende Beispiel, um einen Cluster mit einer 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);
        }
    }
}
```

Rust

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

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

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

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

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

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

    Client::from_conf(config)
}

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

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

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

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

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

```
        .await
        .unwrap();

    cluster_output
}

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

    let cluster = create_cluster(region).await;

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

    Ok(())
}
```

Ruby

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

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

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

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

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

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

def main
  region = "us-east-1"

  cluster = create_cluster(region)

  puts "Created single region cluster:"
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

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

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

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

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

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

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

                return cluster;
            }
        }

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

            var cluster = await Create(region);

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

Golang

Mit folgenden Beispiel erstellen Sie einen Cluster mit einer AWS-Region.

```
package main

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

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

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

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

 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

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

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

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

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

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

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

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

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

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

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

}
```

Abrufen eines Clusters

Die folgenden Beispiele zeigen, wie mit verschiedenen Programmiersprachen Informationen zu einem Cluster mit einer einzelnen Region abrufen.

Python

Verwenden Sie das folgende Beispiel, um Informationen zu einem Cluster mit einer einzelnen Region abzurufen.

```
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 zu einem Cluster mit einer einzelnen Region 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 zu einem Cluster mit einer einzelnen Region 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

Im folgenden Beispiel können Sie Informationen zu einem Cluster mit einer einzigen Region 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 einzigen Region 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 einzigen Region 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 einzigen Region 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 einzigen Region 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

Die folgenden Beispiele veranschaulichen, wie Sie mithilfe verschiedener Programmiersprachen einen Cluster mit einer Region aktualisieren.

Python

Verwenden Sie das folgende Beispiel, um Informationen zu einem Cluster mit einer einzelnen Region 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 einzigen Region 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 Informationen zu einem Cluster mit einer einzelnen Region 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 einzigen Region 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 einzigen Region 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 einzigen Region 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 einzigen Region 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 einzigen Region 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)
 }
}
```

Löschen eines Clusters

Die folgenden Beispiele veranschaulichen, wie Sie mithilfe verschiedener Programmiersprachen einen Cluster mit einer Region löschen.

Python

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

```
import boto3


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

        print("Waiting for cluster to finish deletion")
```

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


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


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

C++

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

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

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

/**
 * Deletes a single-region cluster in Amazon Aurora DSQL
 */
void DeleteCluster(const Aws::String& region, const Aws::String& identifier) {
```

```
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

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

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

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

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

            DeleteCluster(region, clusterId);

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

JavaScript

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

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

async function deleteCluster(region, clusterId) {

  const client = new DSQLClient({ region });

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

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

    await waitUntilClusterNotExists(
      {
        client: client,
        maxWaitTime: 300 // Wait for 5 minutes
      },
      {
        identifier: response.identifier
      }
    );
    console.log(`Cluster Id ${response.identifier} is now deleted`);
    return;
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or already deleted");
    } else {
      console.error("Unable to delete cluster: ", error.message);
    }
    throw error;
  }
}

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

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

main();
```

Java

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

```
package org.example;

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

import java.time.Duration;

public class DeleteCluster {

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


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

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

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
```

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

Rust

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

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

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

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

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

    Client::from_conf(config)
}

/// Delete a DSQL cluster
pub async fn delete_cluster(region: &'static str, identifier: &'static str) {
    let client = dsql_client(region).await;
    let delete_response = client
        .delete_cluster()
```

```
        .identifier(identifier)
        .send()
        .await
        .unwrap();
    println!("Initiated delete of {}", delete_response.arn);

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

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

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

    Ok(())
}
```

Ruby

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

```
require "aws-sdk-dsql"

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

  # The DSQL SDK offers built-in waiters to poll for deletion.
  puts "Waiting for cluster to finish deletion"
  client.wait_until(:cluster_not_exists, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
rescue Aws::Errors::ServiceError => e
  abort "Unable to delete cluster #{identifier} in #{region}: #{e.message}"
end

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

main if $PROGRAM_NAME == __FILE__
```

.NET

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

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

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

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

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

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

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

Golang

Mit folgenden Beispiel löschen Sie einen Cluster mit einer AWS-Region.

```
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)
 }

}
```

Weitere Codebeispiele und Beispiele finden Sie im GitHub-Beispiel-Repository für Aurora DSQL.

Verwenden von AWS-CLI

Die AWS-CLI stellt eine Befehlszeilenschnittstelle für die Verwaltung Ihrer Aurora DSQL-Cluster bereit. In den folgenden Beispielen werden häufig verwendete Cluster-Verwaltungsoperationen demonstriert.

Cluster erstellen

Erstellen eines Clusters unter Verwendung des create-cluster-Befehls.

> **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 er aktiviert wurde.

Example Befehl

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

> **Note**
>
> Geben Sie das `--no-deletion-protection-enabled`-Flag mit an, um den Löschschutz während der Erstellung zu deaktivieren.

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"
    }
}
```

Beschreiben von Clustern

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 mit dem update-cluster-Befehl.

> **Note**
>
> Updates sind asynchrone Vorgänge. Rufen Sie die GetCluster-API auf, bis sich der Status auf `ACTIVE` ändert, 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, für die kein Löschschutz aktiviert ist. Beim Erstellen neuer Cluster ist der Löschschutz standardmäßig aktiviert.

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"
        }
    ]
}
```

Konfigurieren von Clustern mit mehreren Regionen

Konfigurieren und verwalten Sie Cluster für mehrere AWS-Regionen entweder über die AWS CLI oder Ihre bevorzugte Programmiersprache, z. B. Python, C++, JavaScript, Java, Rust, Ruby, .NET und Golang. Die AWS CLI bietet schnellen Zugriff über Shell-Befehle, während die AWS-SDKs eine programmatische Steuerung mit nativer Sprachunterstützung ermöglichen.

Themen

- AWS-SDKs verwenden
- AWS CLI verwenden

AWS-SDKs verwenden

Die AWS-SDKs bieten programmatischen Zugriff auf Aurora DSQL in Ihrer bevorzugten Programmiersprache. In den folgenden Abschnitten sehen Sie, wie allgemeine Clusteroperationen mit verschiedenen Programmiersprachen ausgeführt werden.

Cluster erstellen

In den folgenden Beispielen wird gezeigt, wie ein Cluster mit mehreren Regionen mit verschiedenen Programmiersprachen erstellt wird.

Python

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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={
                'Delay': 10,
                'MaxAttempts': 30
```

```
            }
        )

        return (cluster_1, cluster_2)

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


def main():
    region_1 = "us-east-1"
    region_2 = "us-east-2"
    witness_region = "us-west-2"
    (cluster_1, cluster_2) = create_multi_region_clusters(region_1, region_2,
 witness_region)
    print("Created multi region clusters:")
    print("Cluster id: " + cluster_1['arn'])
    print("Cluster id: " + cluster_2['arn'])


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

C++

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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);

    std::cout << "Creating cluster in " << region1 << std::endl;

    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region
    MultiRegionProperties multiRegionProps1;
    multiRegionProps1.SetWitnessRegion(witnessRegion);
    createClusterRequest1.SetMultiRegionProperties(multiRegionProps1);

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp multi region cluster 1";
    createClusterRequest1.SetTags(tags);
    createClusterRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto createOutcome1 = client1.CreateCluster(createClusterRequest1);
    if (!createOutcome1.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region1 << ": "
                  << createOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster1 = createOutcome1.GetResult();
    std::cout << "Created " << cluster1.GetArn() << std::endl;
```

```
// Create second cluster
std::cout << "Creating cluster in " << region2 << std::endl;

CreateClusterRequest createClusterRequest2;
createClusterRequest2.SetDeletionProtectionEnabled(true);

// Set multi-region properties with witness region and cluster1 as peer
MultiRegionProperties multiRegionProps2;
multiRegionProps2.SetWitnessRegion(witnessRegion);

Aws::Vector<Aws::String> clusters;
clusters.push_back(cluster1.GetArn());
multiRegionProps2.SetClusters(clusters);

tags["Name"] = "cpp multi region cluster 2";
createClusterRequest2.SetMultiRegionProperties(multiRegionProps2);
createClusterRequest2.SetTags(tags);
createClusterRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());

auto createOutcome2 = client2.CreateCluster(createClusterRequest2);
if (!createOutcome2.IsSuccess()) {
    std::cerr << "Failed to create cluster in " << region2 << ": "
              << createOutcome2.GetError().GetMessage() << std::endl;
    throw std::runtime_error("Failed to create multi-region clusters");
}

auto cluster2 = createOutcome2.GetResult();
std::cout << "Created " << cluster2.GetArn() << std::endl;

// Now that we know the cluster2 arn we can set it as a peer of cluster1
UpdateClusterRequest updateClusterRequest;
updateClusterRequest.SetIdentifier(cluster1.GetIdentifier());

MultiRegionProperties updatedProps;
updatedProps.SetWitnessRegion(witnessRegion);

Aws::Vector<Aws::String> updatedClusters;
updatedClusters.push_back(cluster2.GetArn());
updatedProps.SetClusters(updatedClusters);

updateClusterRequest.SetMultiRegionProperties(updatedProps);
updateClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());
```

```
    auto updateOutcome = client1.UpdateCluster(updateClusterRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster in " << region1 << ": "
                  << updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to update multi-region clusters");
    }

    std::cout << "Added " << cluster2.GetArn() << " as a peer of " <<
 cluster1.GetArn() << std::endl;

    return std::make_pair(cluster1, cluster2);
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define regions for the multi-region setup
            Aws::String region1 = "us-east-1";
            Aws::String region2 = "us-east-2";
            Aws::String witnessRegion = "us-west-2";

            auto [cluster1, cluster2] = CreateMultiRegionClusters(region1, region2,
 witnessRegion);

            std::cout << "Created multi region clusters:" << std::endl;
            std::cout << "Cluster 1 ARN: " << cluster1.GetArn() << std::endl;
            std::cout << "Cluster 2 ARN: " << cluster2.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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 updateClusterCommand = new UpdateClusterCommand({
            identifier: response1.identifier,
```

```
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response2.arn]
            }
        });
        await client1.send(updateClusterCommand);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters they'll
 begin the transition to ACTIVE
        console.log(`Waiting for cluster ${response1.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);

        console.log(`Waiting for cluster ${response2.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const region2 = "us-east-2";
```

```
    const witnessRegion = "us-west-2";

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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__
```

Golang

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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)
 }

}
```

.NET

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu erstellen. Das Erstellen von Clustern 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}");
        }
    }
}
```

Abrufen eines Clusters

In den folgenden Beispielen wird gezeigt, wie Informationen zu einem Cluster mit mehreren Regionen mit verschiedenen Programmiersprachen abgerufen werden.

Python

Gehen Sie wie folgt vor, um Informationen zu einem Cluster mit 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 zu einem Cluster mit mehreren Regionen zu erhalten.

```
#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

Gehen Sie wie folgt vor, um Informationen zu einem Cluster mit mehreren Regionen zu erhalten.

```
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

Im folgenden Beispiel wird gezeigt, wie Sie Informationen zu einem Cluster mit 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 wird gezeigt, wie Sie Informationen zu einem Cluster mit 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 wird gezeigt, wie Sie Informationen zu einem Cluster mit 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 wird gezeigt, wie Sie Informationen zu einem Cluster mit 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 wird gezeigt, wie Sie Informationen zu einem Cluster mit 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 Beispielen wird gezeigt, wie ein Cluster mit mehreren Regionen mit verschiedenen Programmiersprachen aktualisiert wird.

Python

Gehen Sie wie folgt vor, um einen Cluster mit 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++

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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

Gehen Sie wie folgt vor, um einen Cluster mit 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)
 }
}
```

Löschen eines Clusters

In den folgenden Beispielen wird gezeigt, wie ein Cluster mit mehreren Regionen mit verschiedenen Programmiersprachen gelöscht wird.

Python

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern mit mehreren Regionen kann einige Zeit in Anspruch nehmen.

```
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++

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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

Gehen Sie wie folgt vor, um einen Cluster mit mehreren Regionen zu löschen. Das Löschen von Clustern 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)
 }
}
```

Weitere Codebeispiele und Beispiele finden Sie im GitHub-Beispiel-Repository für Aurora DSQL.

AWS CLI verwenden

Die AWS CLI bietet eine Befehlszeilenschnittstelle für die Verwaltung Ihrer Aurora DSQL-Cluster mit mehreren Regionen. Die folgenden Beispiele veranschaulichen, wie Sie Cluster mit mehreren Regionen erstellen, konfigurieren und löschen.

Verbindung zu Ihrem Cluster mit mehreren Regionen herstellen

Verbundene Cluster mit mehreren Regionen stellen zwei regionale Endpunkte bereit, jeweils einen in jeder Region des verbundenen AWS-Region-Clusters. Beide Endpunkte stellen eine einzige logische Datenbank dar, die gleichzeitige Lese- und Schreibvorgänge mit hoher Datenkonsistenz unterstützt. Zusätzlich zu den verbundenen Clustern verfügt ein Cluster mit mehreren Regionen auch über eine Witness-Region, die verschlüsselte Transaktionsprotokolle für einen begrenzten Zeitraum speichert. Dies dient der Verbesserung der Beständigkeit und Verfügbarkeit über mehrere Regionen hinweg. Witness-Regionen mit mehreren Regionen haben keine Endpunkte.

Erstellen von Clustern mit mehreren Regionen

Um Cluster mit mehreren Regionen zu erstellen, erstellen Sie zunächst einen Cluster mit einer Witness-Region. Anschließend wird dieser Cluster mit einem zweiten Cluster verbunden, der dieselbe Witness-Region wie Ihr erster Cluster teilt. Im folgenden Beispiel sehen Sie, wie Sie Cluster in USA Ost (Nord-Virginia) und USA Ost (Ohio) mit USA West (Oregon) als Witness-Region erstellen.

Schritt 1: Erstellen Sie Cluster Nummer Eins in USA Ost (Nord-Virginia)

Verwenden Sie den folgenden Befehl, um einen Cluster im Osten der USA (Nord-Virginia) AWS-Region mit multiregionalen Eigenschaften 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 die API-Operation erfolgreich ist, wechselt der Cluster in den `PENDING_SETUP`-Status. Die Clustererstellung verbleibt so lange im Status `PENDING_SETUP`, bis Sie den Cluster mit dem ARN seines Peer-Clusters aktualisieren.

Schritt 2: Erstellen Sie Cluster Nummer Zwei in USA Ost (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 die API-Operation erfolgreich ist, wechselt der Cluster in den `PENDING_SETUP`-Status. Die Clustererstellung verbleibt so lange im Status `PENDING_SETUP`, bis Sie den Cluster mit dem ARN eines anderen Clusters zum Peering aktualisieren.

Schritt 3: Verbinden Sie den Cluster in USA Ost (Nord-Virginia) mit dem in USA Ost (Ohio)

Verwenden Sie den `update-cluster`-Befehl, um Ihr USA Ost (Nord-Virginia)-Cluster mit Ihrem USA Ost (Ohio)-Cluster zu verbinden. Geben Sie den Namen Ihres USA Ost (Nord-Virginia)-Clusters und eine JSON-Zeichenfolge mit dem ARN des USA OSt (Ohio)-Clusters 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: Verbinden Sie den Cluster in USA Ost (Ohio) mit dem in USA Ost (Nord-Virginia)

Verwenden Sie den `update-cluster`-Befehl, um Ihr USA Ost (Nord-Virginia)-Cluster mit Ihrem USA Ost (Ohio)-Cluster zu verbinden. Geben Sie den Namen Ihres USA Ost (Ohio)-Clusters und eine JSON-Zeichenfolge mit dem ARN des USA Ost (Nord-Virginia)-Clusters 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.

Anzeigen von Eigenschaften in Clustern mit mehreren Regionen

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 während der Erstellung

Sie können die Anzahl der Arbeitsschritte 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 mit angeben.

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"]}'
```

So werden die Schritte 2 und 4 kombiniert, aber Sie müssen immer noch Schritt 3 (Aktualisierung des ersten Clusters mit dem ARN des zweiten Clusters) durchführen, um die Peering-Beziehung herzustellen. Nachdem alle Schritte abgeschlossen sind, durchlaufen beide Cluster dieselben

Zustände wie im Standardprozess: von PENDING_SETUP zu CREATING und schließlich zu ACTIVE, wenn sie dann einsatzbereit sind.

Löschen von Clustern mit mehreren Regionen

Es sind zwei Arbeitsschritte erforderlich, um einen Cluster mit mehreren Regionen zu löschen.

1. Deaktivieren Sie den Löschschutz für die einzelnen Cluster.
2. Löschen Sie jeden Peering-Cluster separat in seinem jeweiligen AWS-Region

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

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

```
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 die 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 USA Ost (Ohio) gelöscht haben.

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

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

```
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'
```

Dieser 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 der Validierung versetzt das System beide Peering-Cluster nach einigen Sekunden automatisch in den `DELETING`-Status.

Konfigurieren von Aurora DSQL-Clustern mit AWS CloudFormation

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

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

Erstellen der anfänglichen Clusterkonfiguration

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
```

Nach Cluster-ID suchen

Rufen Sie die physischen Ressourcen-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"
```

```
]
```

Aktualisieren der Clusterkonfiguration

Aktualisieren Sie Ihre AWS CloudFormation-Vorlage so, dass sie beide Cluster-ARNs enthält:

```
---
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
```

Aurora DSQL-Cluster-Lebenszyklus

Wenn Sie den Aurora DSQL-Cluster-Lebenszyklus verstehen, können Sie Ihre Cluster effektiv verwalten. In diesem Abschnitt werden die Definitionen des Cluster-Status und die Funktion „Scale-to-Zero“ zur Kostenoptimierung behandelt.

Definieren des Aurora DSQL-Clusterstatus

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-Managementkonsole Aurora DSQL-API anzeigen. AWS CLI

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.
Aktiv	Der Cluster ist betriebsbereit und einsatzfähig.
Inaktiv	Ein Cluster wird inaktiv, wenn er lange genug inaktiv ist, damit Aurora DSQL die laufenden Ressourcen herunterskalieren kann, um Kapazität und Kosten zu reduzieren. Wenn Sie eine Verbindung zu einem untätigen Cluster herstellen, versetzt Aurora DSQL den Cluster wieder in den Status Aktiv.
Inaktiv	Ein Cluster im Leerlauf wird inaktiv, wenn auf dem Cluster über einen längeren Zeitraum keine Aktivität stattgefunden hat. In diesem angehaltenen Zustand werden die laufenden Ressourcen auf Null skaliert, während Ihre Daten erhalten bleiben. Wenn Sie eine Verbindung zu einem inaktiven Cluster herstellen, versetzt Aurora DSQL den Cluster wieder in den Status Aktiv. Die Dauer der Wiederherstellung hängt von der Clustergröße ab.
Wird aktualisiert	Ein Cluster wechselt in den Status Aktualisierung, wenn Sie Änderungen an der Clusterkonfiguration 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.
Setup steht aus	Nur für Cluster mit mehreren Regionen. Ein Cluster mit mehreren Regionen wechselt in den Status Setup steht aus, wenn Sie in Ihrer ersten Region einen Cluster mit mehreren Regionen mit einer Witness-Region erstellen.

Status	Description
	Die Clustererstellung wird angehalten, bis Sie einen weiteren Cluster in einer sekundären Region erstellen und die beiden Cluster miteinander verbinden.
Löschvorgang steht aus	Nur für Cluster mit mehreren Regionen. Ein Cluster mit mehreren Regionen wechselt in den Status Löschvorgang steht aus, wenn Sie einen Cluster herauslöschen. Der Cluster wechselt in den Status Löschen, nachdem Sie den letzten Peer-Cluster gelöscht haben.

Arbeiten mit Clustern im Leerlauf und Inaktiv

Wenn Aurora DSQL feststellt, dass für einen bestimmten Zeitraum keine Verbindungsaktivität auf einem Cluster besteht, wird der Cluster in den Ruhezustand versetzt, wodurch die laufenden Ressourcen reduziert werden, um Kapazität und Kosten zu minimieren. Wenn die Verbindungsaktivität über einen längeren Zeitraum ausbleibt, wechselt der Cluster im Leerlauf automatisch in den Status Inaktiv, in dem die laufenden Ressourcen auf Null skaliert werden, während Ihre Daten erhalten bleiben.

Um den normalen Betrieb wieder aufzunehmen, stellen Sie einfach wie gewohnt eine Verbindung zum Cluster her. Wenn Sie erfolgreich eine Verbindung zum Cluster herstellen, versetzt Aurora Aurora DSQL den Cluster automatisch in den Status Aktiv.

> **Note**
>
> Der erste Verbindungsversuch mit einem Cluster im Leerlauf oder Inaktiven ist langsamer als gewöhnlich.

Operationen, die einen aktiven Clusterstatus erfordern

Für einige Operationen muss sich Ihr Cluster im Status Aktiv befinden. Um diese Operationen auf einem Cluster im Leerlauf oder Inaktiven auszuführen, müssen Sie Ihren Cluster wieder auf Aktiv umstellen, indem Sie eine Verbindung zu Ihrem Cluster herstellen.

Backup-Operationen

Für die Erstellung eines Backups ist ein aktiver Clusterstatus erforderlich. Wenn sich Ihr Cluster im Leerlauf oder Inaktiv befindet, schlagen Backups mit dem folgenden Fehler fehl:

```
"Error": {
    "Code": "FailedPrecondition",
    "Message": "Cluster 'cluster-id' is in state 'IDLE' and can't be backed up.
    In order to take a backup of your cluster, it must be in Active state. Please
    connect to your cluster to transition it to Active to perform the backup."
}
```

Um mit einer Sicherung fortzufahren:

1. Stellen Sie mithilfe Ihres bevorzugten Datenbankclients oder der Aurora DSQL-Konsole eine Connect zum Cluster her, um ihn aufzuwecken.
2. Warten Sie auf den automatischen Übergang in den Status Aktiv.
3. Initiieren Sie das Backup, nachdem der Cluster voll funktionsfähig ist.

> **Note**
>
> Bestehende Backups, die erstellt wurden, bevor der Cluster inaktiv oder inaktiv war, bleiben gültig und werden nicht beeinträchtigt. Neue Backup-Versuche auf dem Cluster schlagen fehl, bis der Cluster für die auto Aktivierung mit dem Cluster verbunden ist.

Anzeigen Ihres Aurora DSQL-Clusterstatus

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

Konsole

So zeigen Sie den Clusterstatus in der AWS-Managementkonsole an:

Anzeigen des Clusterstatus in der Konsole

1. Öffnen Sie die Aurora DSQL-Konsole unter https://console.aws.amazon.com/dsql.
2. Wählen Sie im Navigationsbereich Clusters (Cluster) aus.
3. Zeigen Sie den Status der einzelnen 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 anzuzeigen.

```
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 an, der gerade gelöscht wird.

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

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 erstellen und verwalten, indem Sie die AWS CLI in einer Befehlszeilen-Shell verwenden. Das AWS CLI bietet direkten Zugriff auf das APIs für AWS-Services, z. B. Aurora DSQL. Syntax und Beispiele für Befehle zu 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.

CloudFormation

Das AWS::DSQL::Clusterist eine CloudFormation Ressource, mit der Sie Aurora DSQL-Cluster als Teil Ihrer Infrastruktur als Code erstellen und verwalten können. 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 CloudFormation Vorlagen verwenden, können Sie Aurora DSQL-Cluster zusammen mit Ihren anderen Cloud-Ressourcen deklarativ bereitstellen. Dadurch wird sichergestellt, dass Ihre Dateninfrastruktur parallel zum Rest Ihres Anwendungsstapels bereitgestellt und verwaltet wird.

Konnektoren für Aurora DSQL

Aurora DSQL bietet spezielle Konnektoren, die bestehende Datenbanktreiber erweitern, um eine nahtlose IAM-Authentifizierung und Integration mit AWS Diensten zu ermöglichen. Diese

Konnektoren sind für Kompatibilität mit gängigen Programmiersprachen und Frameworks konzipiert und gewährleisten auch die Kompatibilität mit vorhandenen PostgreSQL-Workflows.

Zusätzliche Konnektoren sind für future Versionen geplant. Aktuelle Informationen zur Konnektorenverfügbarkeit finden Sie im Aurora DSQL-Beispielrepository.

Verbindung zu Aurora DSQL-Clustern mit einem JDBC-Konnektor herstellen

Der Aurora DSQL Connector für JDBC ist als Authentifizierungs-Plugin konzipiert, das die Funktionalität des PostgreSQL-JDBC-Treibers erweitert, sodass Anwendungen mithilfe von IAM-Anmeldeinformationen bei Aurora DSQL authentifizieren können. Der Konnektor stellt keine direkte Verbindung zur Datenbank her, bietet jedoch eine nahtlose IAM-Authentifizierung zusätzlich zu dem in PostgreSQL zugrunde liegenden JDBC-Treiber.

Der Aurora DSQL Connector für JDBC wurde für die Zusammenarbeit mit dem PostgreSQL JDBC-Treiber entwickelt und bietet eine nahtlose Integration mit den IAM-Authentifizierungsanforderungen von Aurora DSQL.

In Verbindung mit dem PostgreSQL JDBC-Treiber ermöglicht der Aurora DSQL Connector for JDBC die IAM-basierte Authentifizierung für Aurora DSQL. Er ermöglicht eine tiefe Integration mit Authentifizierungsdiensten wie (IAM). AWS AWS Identity and Access Management

Über den Konnektor

Aurora DSQL ist ein verteilter SQL-Datenbankservice, der hohe Verfügbarkeit und Skalierbarkeit für PostgreSQL-kompatible Anwendungen bietet. Aurora DSQL erfordert eine IAM-basierte Authentifizierung mit zeitlich begrenzten Tokens, die vorhandene JDBC-Treiber nicht nativ unterstützen.

Die Hauptidee hinter dem Aurora DSQL Connector for JDBC besteht darin, dem PostgreSQL-JDBC-Treiber eine Authentifizierungsebene hinzuzufügen, die die IAM-Token-Generierung übernimmt, sodass Benutzer eine Verbindung zu Aurora DSQL herstellen können, ohne ihre bestehenden JDBC-Workflows zu ändern.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token

- Token-Generierung: Authentifizierungstoken werden mithilfe von Anmeldeinformationen generiert und haben eine konfigurierbare Lebensdauer AWS

Der Aurora DSQL Connector für JDBC wurde entwickelt, um diese Anforderungen zu verstehen und beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken zu generieren.

Vorteile des Aurora DSQL Connectors für JDBC

Aurora DSQL bietet zwar eine PostgreSQL-kompatible Schnittstelle, jedoch unterstützen vorhandene PostgreSQL-Treiber derzeit nicht die IAM-Authentifizierungsanforderungen von Aurora DSQL. Der Aurora DSQL Connector für JDBC ermöglicht es Kunden, ihre bestehenden PostgreSQL-Workflows weiterhin zu verwenden und gleichzeitig die IAM-Authentifizierung zu aktivieren durch:

- Automatische Token-Generierung: IAM-Token werden automatisch mithilfe von Anmeldeinformationen generiert AWS
- Nahtlose Integration: kompatibel mit vorhandenen JDBC-Verbindungsmustern
- AWS Support für Anmeldeinformationen: Unterstützt verschiedene Anbieter von AWS Anmeldeinformationen (Standard, profilbasiert usw.)

Verwenden des Aurora DSQL Connectors für JDBC mit Verbindungspooling

Der Aurora DSQL Connector für JDBC funktioniert mit Connection Pooling-Bibliotheken wie HikariCP. Der Konnektor übernimmt die Generierung von IAM-Tokens während des Verbindungsaufbaus, sodass die Verbindungspools normal funktionieren können.

Schlüssel-Features

Automatische Tokengenerierung

IAM-Token werden automatisch mithilfe von Anmeldeinformationen generiert. AWS

Nahtlose Integration

Funktioniert mit vorhandenen JDBC-Verbindungsmustern, ohne dass Änderungen im Workflow erforderlich werden.

AWS Support von Anmeldedaten

Unterstützt verschiedene Anbieter von AWS Anmeldeinformationen (standardmäßig, profilbasiert usw.).

Kompatibilität für Verbindungspooling

Nahtlos kompatibel mit Verbindungspooling-Bibliotheken wie HikariCP.

Voraussetzungen

Prüfen Sie bevor Sie beginnen ob die folgenden Voraussetzungen erfüllt sind:

- In Aurora DSQL wurde ein Cluster erstellt.
- Die Java Development Kits (JDK) wurden installiert. Stellen Sie sicher, dass Sie Version 17 oder höher verwenden.
- Richten Sie die entsprechenden IAM-Berechtigungen ein, damit Ihre Anwendung eine Verbindung zu Aurora DSQL herstellen kann.
- AWS konfigurierte Anmeldeinformationen (über AWS CLI Umgebungsvariablen oder IAM-Rollen).

Verwenden des Aurora DSQL Connectors für JDBC

Gehen Sie wie folgt vor, um den Aurora DSQL Connector für JDBC in Ihrer Java-Anwendung zu verwenden:

1. Fügen Sie Ihrem Projekt die folgenden Abhängigkeiten in Ihrem Maven-Projekt hinzu:

```
<dependencies>
    <!-- Aurora DSQL Connector for JDBC -->
    <dependency>
        <groupId>software.amazon.dsql</groupId>
        <artifactId>aurora-dsql-jdbc-connector</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

   Fügen Sie für Gradle-Projekte diese Abhängigkeit hinzu:

```
implementation("software.amazon.dsql:aurora-dsql-jdbc-connector:1.0.0")
```

2. Erstellen Sie eine grundlegende Verbindung zu Ihrem Aurora DSQL-Cluster mithilfe des AWS DSQL PostgreSQL-Connector-Formats:

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DsqlJdbcConnectorExample {
    public static void main(String[] args) {
        // Using AWS DSQL PostgreSQL Connector prefix
        String jdbcUrl = "jdbc:aws-dsql:postgresql://your-cluster.dsql.us-
east-1.on.aws/postgres?user=admin";

        try (Connection connection = DriverManager.getConnection(jdbcUrl)) {
            // Use the connection
            try (Statement statement = connection.createStatement()) {
                // Create a table
                statement.execute("CREATE TABLE IF NOT EXISTS test_table (id UUID
 PRIMARY KEY DEFAULT gen_random_uuid(), name VARCHAR(100))");

                // Insert data
                statement.execute("INSERT INTO test_table (name) VALUES ('Test
 Name')");

                // Query data
                try (ResultSet resultSet = statement.executeQuery("SELECT * FROM
 test_table")) {
                    while (resultSet.next()) {
                        System.out.println("ID: " + resultSet.getInt("id") + ",
 Name: " + resultSet.getString("name"));
                    }
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Konfigurationseigenschaften

Der Aurora DSQL Connector für JDBC unterstützt die folgenden Verbindungseigenschaften:

user

Ermittelt den Benutzer für die Verbindung und die verwendete Methode zur Tokengenerierung. Beispiel: `admin`

token-duration-secs

Gültigkeitsdauer des Tokens in Sekunden. Weitere Informationen zu Token-Limits finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

Profil

Wird für die Instanziierung einer ProfileCredentialsProvider For-Token-Generierung mit dem angegebenen Profilnamen verwendet.

Region

AWS Region für Aurora DSQL-Verbindungen. Es ist optional. Falls angegeben, überschreibt dies die aus der URL extrahierte Region.

Datenbank

Der Name der Datenbank, mit der eine Verbindung hergestellt werden soll. Der Standardwert ist `postgres`.

Protokollierung

Aktivieren Sie die Protokollierung, um Probleme zu beheben, die bei der Verwendung des JDBC-Konnektors für Aurora DSQL auftreten könnten.

Der Konnektor verwendet das integrierte Java-Protokollierungssystem (java.util.logging). Sie können die Protokollierungsebenen konfigurieren, indem Sie eine `logging.properties`-Datei erstellen:

```
# Set root logger level to INFO for clean output
.level = INFO

# Show Aurora DSQL Connector for JDBC FINE logs for detailed debugging
software.amazon.dsql.level = FINE

# Console handler configuration
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

```
# Detailed formatter pattern with timestamp and logger name
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Beispiele

Umfassendere Beispiele und Anwendungsfälle finden Sie im Aurora DSQL Connector for JDBC-Repository

Aurora DSQL-Konnektor für Python

Der Aurora DSQL Connector für Python integriert die IAM-Authentifizierung für die Verbindung von Python-Anwendungen mit Amazon Aurora DSQL-Clustern. Intern verwendet er die Clientbibliotheken psycopg, psycopg2 und asyncpg.

Der Aurora DSQL Connector für Python ist als Authentifizierungs-Plugin konzipiert, das die Funktionalität der Clientbibliotheken psycopg, psycopg2 und asyncpg erweitert, sodass Anwendungen sich mit Amazon Aurora DSQL mithilfe von IAM-Anmeldeinformationen authentifizieren können. Der Connector stellt keine direkte Verbindung zur Datenbank her, sondern bietet zusätzlich zu den zugrunde liegenden Client-Bibliotheken eine nahtlose IAM-Authentifizierung.

Über den Connector

Amazon Aurora DSQL ist ein verteilter SQL-Datenbankservice, der hohe Verfügbarkeit und Skalierbarkeit für PostgreSQL-kompatible Anwendungen bietet. Aurora DSQL erfordert eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token, die bestehende Python-Bibliotheken nicht nativ unterstützen.

Die Idee hinter dem Aurora DSQL Connector für Python besteht darin, zusätzlich zu den Clientbibliotheken psycopg, psycopg2 und asyncpg eine Authentifizierungsebene hinzuzufügen, die die IAM-Token-Generierung übernimmt und es Benutzern ermöglicht, sich mit Aurora DSQL zu verbinden, ohne ihre bestehenden Workflows zu ändern.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token
- Token-Generierung: Authentifizierungstoken werden mithilfe von AWS-Anmeldeinformationen generiert und haben konfigurierbare Lebensdauern

Der Aurora DSQL Connector für Python wurde entwickelt, um diese Anforderungen zu verstehen und beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken zu generieren.

Features

- Automatische IAM-Authentifizierung — IAM-Token werden automatisch mithilfe von AWS-Anmeldeinformationen generiert
- Basiert auf psycopg, psycopg2 und asyncpg — nutzt die Clientbibliotheken psycopg, psycopg2 und asyncpg
- Nahtlose Integration — Funktioniert mit bestehenden Verbindungsmustern von psycopg, psycopg2 und asyncpg, ohne dass Änderungen am Arbeitsablauf erforderlich sind
- Automatische Erkennung der Region — Extrahiert die AWS-Region aus dem DSQL-Cluster-Hostnamen
- Support für AWS-Anmeldeinformationen — Unterstützt verschiedene AWS-Anmeldeinformationsanbieter (Standard, profilbasiert, benutzerdefiniert)
- Kompatibilität mit Verbindungspooling — Funktioniert mit dem integrierten Verbindungspooling von psycopg, psycopg2 und asyncpg

Schnellstart-Anleitung

Voraussetzungen

- Python 3.10 oder höher
- Zugriff auf einen Aurora DSQL-Cluster
- Richten Sie die entsprechenden IAM-Berechtigungen ein, damit Ihre Anwendung eine Verbindung zu Aurora DSQL herstellen kann.
- AWS-Anmeldeinformationen konfiguriert (über AWS-CLI, Umgebungsvariablen oder IAM-Rollen)

Installation

```
pip install aurora-dsql-python-connector
```

Installieren Sie psycopg oder psycopg2 oder asyncpg separat

Das Aurora DSQL Connector for Python-Installationsprogramm installiert die zugrunde liegenden Bibliotheken nicht. Sie müssen separat installiert werden, z. B.:

```
# Install psycopg and psycopg pool
```

```
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
pip install psycopg2-binary
```

```
# Install asyncpg
pip install asyncpg
```

Hinweis:

Nur die Bibliothek, die benötigt wird, muss installiert werden. Wenn der Client psycopg verwenden möchte, muss daher nur psycopg installiert werden. Wenn der Client psycopg2 verwenden soll, muss nur psycopg2 installiert werden. Wenn der Client asyncpg verwenden soll, muss nur asyncpg installiert werden.

Wenn der Client mehr als eine benötigt, müssen alle benötigten Bibliotheken installiert werden.

Grundlegende Verwendung

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
```

```
    'user': "admin",
}

conn = dsql.connect(**config)
with conn.cursor() as cur:
    cur.execute("SELECT 1")
    result = cur.fetchone()
    print(result)
```

asyncpg

```
import asyncio
import aurora_dsql_asyncpg as dsql

config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin",
}

conn = await dsql.connect(**config)
result = await conn.fetchrow("SELECT 1")
await conn.close()
print(result)
```

Benutze nur den Host

psycopg

```
import aurora_dsql_psycopg as dsql

conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

psycopg2

```
import aurora_dsql_psycopg2 as dsql

conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

asyncpg

```
import asyncio
```

```
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

Nur Cluster-ID verwenden

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("your-cluster")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster")
```

Hinweis:

Im Szenario „Nur Cluster-ID verwenden“ wird die Region verwendet, die zuvor auf dem Computer festgelegt wurde, z. B.:

```
aws configure set region us-east-1
```

Wenn die Region nicht festgelegt wurde oder sich die angegebene Cluster-ID in einer anderen Region befindet, schlägt die Verbindung fehl. Damit es funktioniert, geben Sie Region als Parameter an, wie im folgenden Beispiel:

psycopg

```
    import aurora_dsql_psycopg as dsql
```

```
    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
            "region": "us-east-1",
    }

    conn = await dsql.connect("your-cluster", **config)
```

Verbindungszeichenfolge

Psychopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql
```

```
    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/
postgres?user=admin&token_duration_secs=15")
```

Erweiterte Konfiguration

Psychopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
```

```
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Konfigurationsoptionen

Option	Typ	Erforderlich	Beschreibung
`host`	`string`	Ja	Hostname oder Cluster-ID des DSQL-Clusters
`user`	`string`	Nein	DSQL-Benutzername. Standard: admin
`dbname`	`string`	Nein	Datenbankname. Standard: Postgres
`region`	`string`	Nein	AWS-Region (automatisch anhand des Hostnamens erkannt, falls nicht angegeben)
`port`	`int`	Nein	Die Standardeinstellung ist 5432

Option	Typ	Erforderlich	Beschreibung
`custom_credentials_provider`	`CredentialProvider`	Nein	Anbieter für benutzerdefinierte AWS-Anmeldeinformationen
`profile`	`string`	Nein	Der Name des IAM-Profils. Standard: Standard.
`token_duration_secs`	`int`	Nein	Ablaufzeit des Tokens in Sekunden

Alle Standardverbindungsoptionen der zugrunde liegenden Bibliotheken psycopg, psycopg2 und asyncpg werden ebenfalls unterstützt, mit Ausnahme der asyncpg-Parameter krbsrvname und gsslib, die von DSQL nicht unterstützt werden.

Verwenden des Aurora DSQL-Konnektors für Python mit Verbindungspooling

Der Aurora DSQL Connector für Python funktioniert mit dem integrierten Verbindungspooling von psycopg, psycopg2 und asyncpg. Der Konnektor übernimmt die Generierung von IAM-Tokens während des Verbindungsaufbaus, sodass die Verbindungspools normal funktionieren können.

psycopg

Für psycopg implementiert der Konnektor eine Verbindungsklasse namens DSQLConnection , die direkt an den psycopg_pool übergeben werden kann. ConnectionPool Konstruktor. Für asynchrone Operationen gibt es auch eine asynchrone Version der Klasse namens Connection. DSQLAsync

```
    from psycopg_pool import ConnectionPool as PsycopgPool

    ...
    pool = PsycopgPool(
        "",
        connection_class=dsql.DSQLConnection,
        kwargs=conn_params,
        min_size=2,
        max_size=8,
        max_lifetime=3300
    )
```

Hinweis: Max_Lifetime-Konfiguration der Verbindung

Der Parameter max_lifetime sollte auf weniger als 3600 Sekunden (eine Stunde) gesetzt werden, da dies die maximale Verbindungsdauer ist, die von der Aurora DSQL-Datenbank zulässig ist. Wenn Sie einen niedrigeren Wert für max_lifetime festlegen, kann der Verbindungspool die Wiederverwendung von Verbindungen proaktiv verwalten. Dies ist effizienter als die Behandlung von Verbindungs-Timeout-Fehlern in der Datenbank.

psycopg2

Für psycopg2 stellt der Konnektor eine Klasse namens Aurora bereit, die von DSQLThreaded ConnectionPool psycopg2.pool erbt. ThreadedConnectionPool. Die DSQLThreaded ConnectionPool Aurora-Klasse überschreibt nur die interne _connect-Methode. Der Rest der Implementierung wird von psycopg2.pool bereitgestellt. ThreadedConnectionPool unverändert.

```
    import aurora_dsql_psycopg2 as dsql

    pool = dsql.AuroraDSQLThreadedConnectionPool(
            minconn=2,
            maxconn=8,
            **conn_params,
    )
```

asyncpg

Für asyncpg stellt der Konnektor eine create_pool-Funktion bereit, die eine Instanz von asyncPG.Pool zurückgibt.

```
    import asyncio
    import os

    import aurora_dsql_asyncpg as dsql

    pool_params = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'user': "admin",
        "min_size": 2,
        "max_size": 5,
    }

    pool = await dsql.create_pool(**pool_params)
```

Authentifizierung

Der Connector verarbeitet die DSQL-Authentifizierung automatisch, indem er mithilfe des DSQL-Client-Token-Generators Token generiert. Wenn die AWS-Region nicht angegeben wird, wird sie automatisch anhand des angegebenen Hostnamens analysiert.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie im Benutzerhandbuch.

Admin im Vergleich zu regulären Benutzern

- Benannte Benutzer verwenden `"admin"` automatisch Administratorauthentifizierungstoken
- Alle anderen Benutzer verwenden Authentifizierungstoken, die keine Administratorrechte haben
- Token werden dynamisch für jede Verbindung generiert

Beispiele

Den vollständigen Beispielcode finden Sie in den Beispielen, die in den folgenden Abschnitten angegeben sind. Anweisungen zur Ausführung der Beispiele finden Sie in den READMDE-Beispieldateien.

psycopg

Beispiele README

Description	Beispiele
Verwenden des Aurora DSQL Connectors für Python für grundlegende Verbindungen	Beispiel für eine grundlegende Verbindung
Verwenden des Aurora DSQL Connectors für Python für grundlegende asynchrone Verbindungen	Grundlegendes Beispiel für eine asynchrone Verbindung
Verwenden des Aurora DSQL Connectors für	Einfaches Verbindungsbeispiel mit Verbindungspool

Description	Beispiele
Python mit Verbindungspool	
	Beispiel für gleichzeitige Verbindungen mit Verbindungspool
Verwenden des Aurora DSQL Connectors für Python mit asynchronem Verbindungspool	Einfaches Verbindungsbeispiel mit asynchronem Verbindungspool

psycopg2

Beispiele README

Description	Beispiele
Verwenden des Aurora DSQL Connectors für Python für grundlegende Verbindungen	Beispiel für eine grundlegende Verbindung
Verwenden des Aurora DSQL Connectors für Python mit Verbindungspool	Einfaches Verbindungsbeispiel mit Verbindungspool
	Beispiel für gleichzeitige Verbindungen mit Verbindungspool

asyncpg

Beispiele README

Description	Beispiele
Verwenden des Aurora DSQL Connectors für	Beispiel für eine grundlegende Verbindung

Description	Beispiele
Python für grundlegende Verbindungen	
Verwenden des Aurora DSQL Connectors für Python mit Verbindungspool	Einfaches Verbindungsbeispiel mit Verbindungspool
	Beispiel für gleichzeitige Verbindungen mit Verbindungspool

Verbindung zu Aurora DSQL-Clustern mit einem Go-Connector herstellen

Der Aurora DSQL Connector for Go umschließt PGX mit automatischer IAM-Authentifizierung. Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und das Verbindungsmanagement, sodass Sie sich auf Ihre Anwendungslogik konzentrieren können.

Über den Konnektor

Aurora DSQL erfordert eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token, die bestehende Go PostgreSQL-Treiber nicht nativ unterstützen. Der Aurora DSQL Connector for Go fügt dem pgx-Treiber eine Authentifizierungsebene hinzu, die die IAM-Token-Generierung übernimmt, sodass Sie eine Verbindung zu Aurora DSQL herstellen können, ohne Ihre bestehenden pgx-Workflows zu ändern.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token
- Token-Generierung: Der Connector generiert Authentifizierungstoken mithilfe von AWS Anmeldeinformationen, und diese Token haben eine konfigurierbare Lebensdauer

Der Aurora DSQL Connector for Go wurde entwickelt, um diese Anforderungen zu verstehen und beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken zu generieren.

Vorteile des Aurora DSQL Connector for Go

Mit dem Aurora DSQL Connector for Go können Sie Ihre bestehenden PGX-Workflows weiterhin verwenden und gleichzeitig die IAM-Authentifizierung aktivieren über:

- Automatische Token-Generierung: Der Connector generiert automatisch IAM-Token für jede Verbindung
- Verbindungspooling: Integrierte Unterstützung für die `pgxpool` automatische Token-Generierung pro Verbindung
- Flexible Konfiguration: Support für vollständige Endgeräte oder Cluster IDs mit automatischer Regionserkennung
- AWS Support für Anmeldeinformationen: Unterstützt AWS Profile und Anbieter von benutzerdefinierten Anmeldeinformationen

Schlüssel-Features

Automatische Tokenverwaltung

Der Connector generiert automatisch IAM-Token für jede neue Verbindung unter Verwendung vorab aufgelöster Anmeldeinformationen.

Verbindungspooling

Verbindungspooling über `pgxpool` mit automatischer Token-Generierung pro Verbindung.

Flexible Host-Konfiguration

Unterstützt sowohl vollständige Cluster-Endpunkte als auch Cluster IDs mit automatischer Regionserkennung.

SSL-Sicherheit

SSL ist im Modus „Verify-Full“ und direkter TLS-Aushandlung immer aktiviert.

Voraussetzungen

- Go 1.24 oder höher
- AWS Anmeldeinformationen konfiguriert
- Ein Aurora DSQL-Cluster

Der Connector verwendet die standardmäßige Anmeldeinformationskette des AWS SDK for Go v2, die Anmeldeinformationen in der folgenden Reihenfolge auflöst:

1. Umgebungsvariablen (AWS_ACCESS_KEY_ID, _ACCESS_KEY) AWS_SECRET
2. Datei mit gemeinsam genutzten Anmeldeinformationen (~/.aws/credentials)
3. Gemeinsam genutzte Konfigurationsdatei (~/.aws/config)
4. IAM-Rolle für Amazon EC2/ECS/Lambda

Installation

Installieren Sie den Connector mithilfe von Go-Modulen:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql
```

Schnellstart

Das folgende Beispiel zeigt, wie Sie einen Verbindungspool erstellen und eine Abfrage ausführen:

```
package main

import (
    "context"
    "log"

    "github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql"
)

func main() {
    ctx := context.Background()

    // Create a connection pool
    pool, err := dsql.NewPool(ctx, dsql.Config{
        Host: "your-cluster.dsql.us-east-1.on.aws",
    })
    if err != nil {
        log.Fatal(err)
    }
    defer pool.Close()

    // Execute a query
```

```
    var greeting string
    err = pool.QueryRow(ctx, "SELECT 'Hello, DSQL!'").Scan(&greeting)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(greeting)
}
```

Konfigurationsoptionen

Der Connector unterstützt die folgenden Konfigurationsoptionen:

Feld	Typ	Standard	Description
Host	Zeichenfolge	(erforderlich)	Cluster-Endpunkt oder Cluster-ID
Region	Zeichenfolge	(automatisch erkannt)	AWS Region; erforderlich, wenn Host eine Cluster-ID ist
Benutzer	Zeichenfolge	„Administrator“	Datenbankbenutzer
Datenbank	Zeichenfolge	„Postgres“	Datenbankname
Port	int	5432	Datenbankport
Profil	Zeichenfolge	""	AWS Profilname für Anmeldeinformationen
TokenDurationSecs	int	900 (15 Minuten)	Gültigkeitsdauer des Tokens in Sekunden (maximal zulässig: 1 Woche, Standard: 15 Minuten)
MaxConns	int32	0	Maximale Anzahl an Pool-Verbindungen (0 = pgxpool-Standard)
MinConns	int32	0	Minimale Anzahl an Poolverbindungen (0 = pgxpool-Standard)
MaxConnLifetime	Zeit.Dauer	55 Minuten	Maximale Verbindungslebensdauer

Format der Verbindungszeichenfolge

Der Connector unterstützt PostgreSQL- und DSQL-Verbindungszeichenfolgenformate:

```
postgres://[user@]host[:port]/[database][?param=value&...]
dsql://[user@]host[:port]/[database][?param=value&...]
```

Unterstützte Abfrageparameter:

- `region`- AWS Region
- `profile`- AWS Profilname
- `tokenDurationSecs`- Gültigkeitsdauer des Tokens in Sekunden

Beispiele:

```
// Full endpoint (region auto-detected)
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres")

// Using dsql:// scheme (also supported)
pool, _ := dsql.NewPool(ctx, "dsql://admin@cluster.dsql.us-east-1.on.aws/postgres")

// With explicit region
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/mydb?
region=us-east-1")

// With AWS profile
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?
profile=dev")
```

Fortgeschrittene Nutzung

Host-Konfiguration

Der Connector unterstützt zwei Host-Formate:

Vollständiger Endpunkt (Region automatisch erkannt):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
```

```
})
```

Cluster-ID (Region erforderlich):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host:   "your-cluster-id",
    Region: "us-east-1",
})
```

Optimierung der Poolkonfiguration

Konfigurieren Sie den Verbindungspool für Ihren Workload:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:              "your-cluster.dsql.us-east-1.on.aws",
    MaxConns:          20,
    MinConns:          5,
    MaxConnLifetime:   time.Hour,
    MaxConnIdleTime:   30 * time.Minute,
    HealthCheckPeriod: time.Minute,
})
```

Nutzung einer einzelnen Verbindung

Für einfache Skripte oder wenn kein Verbindungspooling benötigt wird:

```
conn, err := dsql.Connect(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
})
if err != nil {
    log.Fatal(err)
}
defer conn.Close(ctx)

// Use the connection
rows, err := conn.Query(ctx, "SELECT * FROM users")
```

Profile verwenden AWS

Geben Sie ein AWS Profil für Anmeldeinformationen an:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
```

```
    Host:    "your-cluster.dsql.us-east-1.on.aws",
    Profile: "production",
})
```

OCC-Wiederholung

Aurora DSQL verwendet Optimistic Concurrency Control (OCC). Wenn zwei Transaktionen dieselben Daten ändern, gewinnt die erste Transaktion, die einen Commit durchführt, und die zweite erhält einen OCC-Fehler.

Das `occretry` Paket bietet Hilfsprogramme für automatische Wiederholungen bei exponentiellem Backoff und Jitter. Installiere es mit:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/occretry
```

`WithRetry`Für transaktionale Schreibvorgänge verwenden:

```
err := occretry.WithRetry(ctx, pool, occretry.DefaultConfig(), func(tx pgx.Tx) error {
    _, err := tx.Exec(ctx, "UPDATE accounts SET balance = balance - $1 WHERE id = $2",
 100, fromID)
    if err != nil {
        return err
    }
    _, err = tx.Exec(ctx, "UPDATE accounts SET balance = balance + $1 WHERE id = $2",
 100, toID)
    return err
})
```

Verwenden Sie für DDL- oder Einzelanweisungen: `ExecWithRetry`

```
err := occretry.ExecWithRetry(ctx, pool, occretry.DefaultConfig(),
    "CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY, name TEXT)")
```

> **Important**
>
> `WithRetry`verwaltet`BEGIN`/`COMMIT`/`ROLLBACK`intern. Ihr Callback empfängt eine Transaktion und sollte nur Datenbankoperationen enthalten und sicher sein, es erneut zu versuchen.

Beispiele

Umfassendere Beispiele und Anwendungsfälle finden Sie in den Aurora DSQL Connector for Go-Beispielen.

Beispiel	Description
example_preferred	Empfohlen: Verbindungspool mit gleichzeitigen Abfragen
Transaktion	Transaktionsabwicklung mit BEGIN/COMMIT/ROLLBACK
occ_retry	Umgang mit OCC-Konflikten mit exponentiellem Backoff
Verbindungszeichenfolge	Verbindungszeichenfolgen für die Konfiguration verwenden

Aurora DSQL-Konnektoren für Node.js

Der Aurora DSQL Connector für node-postgres und der Aurora DSQL Connector für Postgres.js sind Authentifizierungs-Plugins, die die Funktionalität der Clients node-postgres und Postgres.js erweitern, sodass Anwendungen sich mit Aurora DSQL mithilfe von IAM-Anmeldeinformationen authentifizieren können.

Aurora DSQL-Konnektor für Node-Postgres

Der Aurora DSQL Connector für Node-Postgres ist ein Node.js Connector, der auf Node-Postgres basiert und die IAM-Authentifizierung für die Verbindung von Anwendungen mit Amazon Aurora DSQL-Clustern integriert. JavaScript/TypeScript

Der Aurora DSQL Connector ist als Authentifizierungs-Plugin konzipiert, das die Funktionalität des Clients und Pools von Node-Postgres erweitert, sodass Anwendungen sich mit Amazon Aurora DSQL mithilfe von IAM-Anmeldeinformationen authentifizieren können.

Über den Connector

Amazon Aurora DSQL ist eine Cloud-native verteilte Datenbank mit PostgreSQL-Kompatibilität. Sie erfordert zwar IAM-Authentifizierung und zeitgebundene Token, aber bei herkömmlichen Node.js Datenbanktreibern fehlt diese integrierte Unterstützung.

Der Aurora DSQL Connector für Node-Postgres schließt diese Lücke, indem er eine Authentifizierungs-Middleware implementiert, die nahtlos mit Node-Postgres zusammenarbeitet.

Dieser Ansatz ermöglicht es Entwicklern, ihren bestehenden Node-Postgres-Code beizubehalten und gleichzeitig durch automatisiertes Token-Management sicheren IAM-basierten Zugriff auf Aurora DSQL-Cluster zu erhalten.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token
- Token-Generierung: Authentifizierungstoken werden mithilfe von AWS-Anmeldeinformationen generiert und haben eine konfigurierbare Lebensdauer

Der Aurora DSQL Connector für Node-Postgres wurde entwickelt, um diese Anforderungen zu verstehen und beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken zu generieren.

Features

- Automatische IAM-Authentifizierung — Übernimmt die Generierung und Aktualisierung von DSQL-Tokens
- Basiert auf Node-Postgres — Nutzt den beliebten PostgreSQL-Client für Node.js
- Nahtlose Integration — Funktioniert mit bestehenden Node-Postgres-Verbindungsmustern
- Automatische Erkennung der Region — Extrahiert die AWS-Region aus dem DSQL-Cluster-Hostnamen
- TypeScript Umfassender Support — bietet volle Typsicherheit
- Support für AWS-Anmeldeinformationen — Unterstützt verschiedene AWS-Anmeldeinformationsanbieter (Standard, profilbasiert, benutzerdefiniert)
- Kompatibilität mit Verbindungspooling — Funktioniert problemlos mit dem integrierten Verbindungspooling

Beispielanwendung

Im Beispiel ist eine Beispielanwendung enthalten, die zeigt, wie Aurora DSQL Connector für Node-Postgres verwendet wird. Um das mitgelieferte Beispiel auszuführen, lesen Sie bitte die README-Beispieldatei.

Schnellstartanleitung

Voraussetzungen

- Node.js 20+
- Zugriff auf einen Aurora DSQL-Cluster
- Richten Sie die entsprechenden IAM-Berechtigungen ein, damit Ihre Anwendung eine Verbindung zu Aurora DSQL herstellen kann.
- AWS-Anmeldeinformationen konfiguriert (über AWS-CLI, Umgebungsvariablen oder IAM-Rollen)

Installation

```
npm install @aws/aurora-dsql-node-postgres-connector
```

Abhängigkeiten von Kollegen

```
npm install @aws-sdk/credential-providers @aws-sdk/dsql-signer pg tsx
npm install --save-dev @types/pg
```

Usage

Client-Verbindung

```
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
});
await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Pool-Verbindung

```
import { AuroraDSQLPool } from "@aws/aurora-dsql-node-postgres-connector";

const pool = new AuroraDSQLPool({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
  max: 3,
```

```
  idleTimeoutMillis: 60000,
});

const result = await pool.query("SELECT NOW()");
```

Erweiterte Nutzung

```
import { fromNodeProviderChain } from "@aws-sdk/credential-providers";
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "example.dsql.us-east-1.on.aws",
  user: "admin",
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
});

await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Konfigurationsoptionen

Option	Typ	Erfo ich	Beschreibung
`host`	`string`	Ja	Hostname des DSQL-Clusters
`username`	`string`	Ja	DSQL-Benutzername
`database`	`string`	Nei	Datenbankname
`region`	`string`	Nei	AWS-Region (automatisch anhand des Hostnamens erkannt, falls nicht angegeben)
`port`	`number`	Nei	Die Standardeinstellung ist 5432
`customCredentialsProvider`	`AwsCredentialIdentity / AwsCredentialIdentityProvider`	Nei	Anbieter für benutzerdefinierte AWS-Anmeldeinformationen

Option	Typ	Erfo ich	Beschreibung
`profile`	`string`	Nei	Der IAM-Profilname. Die Standarde instellung ist „Standard“
`tokenDura tionSecs`	`number`	Nei	Ablaufzeit des Tokens in Sekunden

Alle anderen Parameter von Client /Pool werden unterstützt.

Authentifizierung

Der Connector verarbeitet die DSQL-Authentifizierung automatisch, indem er mithilfe des DSQL-Client-Token-Generators Token generiert. Wenn die AWS-Region nicht angegeben wird, wird sie automatisch anhand des angegebenen Hostnamens analysiert.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie im Benutzerhandbuch.

Admin im Vergleich zu regulären Benutzern

- Benutzer mit dem Namen „admin“ verwenden automatisch Admin-Authentifizierungstoken
- Alle anderen Benutzer verwenden reguläre Authentifizierungstoken
- Token werden dynamisch für jede Verbindung generiert

Aurora DSQL-Konnektor für Postgres.js

Der Aurora DSQL Connector for Postgres.js ist ein Node.js Connector, der auf Postgres.js basiert und die IAM-Authentifizierung für die Verbindung von JavaScript Anwendungen mit Amazon Aurora DSQL-Clustern integriert.

Der Aurora DSQL Connector für Postgres.js ist als Authentifizierungs-Plugin konzipiert, das die Funktionalität des Postgres.js -Clients erweitert, sodass Anwendungen sich mit Amazon Aurora DSQL mithilfe von IAM-Anmeldeinformationen authentifizieren können. Der Connector stellt keine direkte Verbindung zur Datenbank her, sondern bietet zusätzlich zum zugrunde liegenden Postgres.js -Treiber eine nahtlose IAM-Authentifizierung.

Über den Connector

Amazon Aurora DSQL ist ein verteilter SQL-Datenbankservice, der hohe Verfügbarkeit und Skalierbarkeit für PostgreSQL-kompatible Anwendungen bietet. Aurora DSQL erfordert eine IAM-basierte Authentifizierung mit zeitlich begrenzten Tokens, die vorhandene Node.js -Treiber nicht nativ unterstützen.

Die Idee hinter dem Aurora DSQL Connector for Postgres.js besteht darin, dem Client Postgres.js eine Authentifizierungsebene hinzuzufügen, die die Generierung von IAM-Tokens übernimmt, sodass Benutzer eine Verbindung zu Aurora DSQL herstellen können, ohne ihre bestehenden Postgres.js -Workflows zu ändern.

Der Aurora DSQL Connector für Postgres.js funktioniert mit den meisten Versionen von Postgres.js. Benutzer stellen ihre eigene Version bereit, indem sie Postgres.js direkt installieren.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token
- Token-Generierung: Authentifizierungstoken werden mithilfe von AWS-Anmeldeinformationen generiert und haben konfigurierbare Lebensdauern

Der Aurora DSQL Connector für Postgres.js wurde entwickelt, um diese Anforderungen zu verstehen und beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken zu generieren.

Features

- Automatische IAM-Authentifizierung — Übernimmt die Generierung und Aktualisierung von DSQL-Tokens
- Basiert auf Postgres.js — Nutzt den schnellen PostgreSQL-Client für Node.js
- Nahtlose Integration — Funktioniert mit vorhandenen Postgres.js Verbindungsmustern
- Automatische Erkennung der Region — Extrahiert die AWS-Region aus dem DSQL-Cluster-Hostnamen
- TypeScript Umfassender Support — bietet volle Typsicherheit
- Support für AWS-Anmeldeinformationen — Unterstützt verschiedene AWS-Anmeldeinformationsanbieter (Standard, profilbasiert, benutzerdefiniert)

- Kompatibilität mit Verbindungspooling — Funktioniert nahtlos mit dem integrierten Verbindungspooling von Postgres.js

Schnellstart-Anleitung

Voraussetzungen

- Node.js 20+
- Zugriff auf einen Aurora DSQL-Cluster
- Richten Sie die entsprechenden IAM-Berechtigungen ein, damit Ihre Anwendung eine Verbindung zu Aurora DSQL herstellen kann.
- AWS-Anmeldeinformationen konfiguriert (über AWS-CLI, Umgebungsvariablen oder IAM-Rollen)

Installation

```
npm install @aws/aurora-dsql-postgresjs-connector
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Grundlegende Verwendung

```
import { auroraDSQLPostgres } from '@aws/aurora-dsql-postgresjs-connector';

const sql = auroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  username: 'admin',

});

// Execute queries
const result = await sql`SELECT current_timestamp`;
console.log(result);

// Clean up
await sql.end();
```

Cluster-ID anstelle von Host verwenden

```
const sql = auroraDSQLPostgres({
```

```
  host: 'your-cluster-id',
  region: 'us-east-1',
  username: 'admin',

});
```

Verbindungszeichenfolge

```
const sql = AuroraDSQLPostgres(
  'postgres://admin@your-cluster.dsql.us-east-1.on.aws'
);

const result = await sql`SELECT current_timestamp`;
```

Erweiterte Konfiguration

```
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';

const sql = AuroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  database: 'postgres',
  username: 'admin',
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
  tokenDurationSecs: 3600,                            // Token expiration (seconds)

  // Standard Postgres.js options
  max: 20,                                // Connection pool size
  ssl: { rejectUnauthorized: false }    // SSL configuration
});
```

Konfigurationsoptionen

Option	Typ	Erforderlich	Beschreibung
`host`	`string`	Ja	Hostname oder Cluster-ID des DSQL-Clusters
`database`	`string?`	Nein	Datenbankname

Option	Typ	Erforderlich	Beschreibung
`username`	`string?`	Nein	Datenbank-Benutzername (verwendet admin, falls nicht angegeben)
`region`	`string?`	Nein	AWS-Region (automatisch anhand des Hostnamens erkannt, falls nicht angegeben)
`customCredentialsProvider`	`AwsCredentialIdentityProvider?`	Nein	Anbieter für benutzerdefinierte AWS-Anmeldeinformationen
`tokenDurationSecs`	`number?`	Nein	Ablaufzeit des Tokens in Sekunden

Alle Standardoptionen von Postgres.js werden ebenfalls unterstützt.

Authentifizierung

Der Connector verarbeitet die DSQL-Authentifizierung automatisch, indem er mithilfe des DSQL-Client-Token-Generators Token generiert. Wenn die AWS-Region nicht angegeben wird, wird sie automatisch anhand des angegebenen Hostnamens analysiert.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie im Benutzerhandbuch.

Admin im Vergleich zu regulären Benutzern

- Benutzer mit dem Namen „admin“ verwenden automatisch Admin-Authentifizierungstoken
- Alle anderen Benutzer verwenden reguläre Authentifizierungstoken
- Tokens werden dynamisch für jede Verbindung generiert

Beispiel für die Verwendung

JavaScript Beispiele, die den Aurora DSQL Connector für Postgres.js verwenden, finden Sie unter GitHub. Anweisungen zur Ausführung der Beispiele finden Sie im Beispielverzeichnis.

Description	Beispiel
Verbindungspooling mit gleichzeitigen Abfragen, einschließlich Tabellene rstellung, Einfügungen und Lesevorgängen für mehrere Worker	Beispiel für einen Verbindungspool (bevorzugt)
CRUD-Operationen (Tabelle erstellen, einfügen, auswählen, löschen) ohne Verbindungspooling	Beispiel ohne Verbindungspool

Verbindung zu Aurora DSQL-Clustern mit einem Ruby-Konnektor herstellen

Der Aurora DSQL Connector für Ruby ist ein auf pg basierender Ruby-Connector, der die IAM-Authentifizierung für die Verbindung von Ruby-Anwendungen mit Amazon Aurora DSQL-Clustern integriert.

Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und das Verbindungspooling, sodass Sie sich auf Ihre Anwendungslogik konzentrieren können.

Über den Konnektor

Amazon Aurora DSQL erfordert eine IAM-Authentifizierung mit zeitlich begrenzten Token, die von vorhandenen Ruby PostgreSQL-Treibern nicht nativ unterstützt werden. Der Aurora DSQL Connector für Ruby fügt dem PG-Gem eine Authentifizierungsebene hinzu, die die IAM-Token-Generierung übernimmt, sodass Sie eine Verbindung zu Aurora DSQL herstellen können, ohne Ihre bestehenden pg-Workflows zu ändern.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token

- Token-Generierung: Der Connector generiert Authentifizierungstoken mithilfe von AWS Anmeldeinformationen, und diese Token haben eine konfigurierbare Lebensdauer

Der Aurora DSQL Connector for Ruby versteht diese Anforderungen und generiert beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken.

Features

- Automatische IAM-Authentifizierung — Übernimmt die Generierung und Aktualisierung von Aurora DSQL-Tokens
- Built on pg — Umschließt das beliebte PostgreSQL-Gem für Ruby
- Nahtlose Integration — Funktioniert mit bestehenden PG-Gem-Workflows
- Verbindungspooling — Integrierte Unterstützung über das `connection_pool` Gem mit max_lifetime-Durchsetzung
- Automatische Erkennung von Regionen — Extrahiert die AWS Region aus dem Hostnamen des Aurora DSQL-Clusters
- AWS Unterstützung für Anmeldeinformationen — Unterstützt AWS Profile und Anbieter von benutzerdefinierten Anmeldeinformationen
- OCC-Wiederholung — Opt-In für optimistische Parallelitätssteuerung mit exponentiellem Backoff

Beispiel

Ein vollständiges Beispiel finden Sie in der Beispielanwendung unter. GitHub

Schnellstart-Anleitung

Voraussetzungen

- Ruby 3.1 oder höher
- Zugriff auf einen Aurora DSQL-Cluster
- AWS konfigurierte Anmeldeinformationen (über AWS CLI, Umgebungsvariablen oder IAM-Rollen)

Installation

Fügen Sie zu Ihrem Gemfile hinzu:

```
gem "aurora-dsql-ruby-pg"
```

Oder direkt installieren:

```
gem install aurora-dsql-ruby-pg
```

Usage

Verbindung zum Pool

```
require "aurora_dsql_pg"

# Create a connection pool with OCC retry enabled
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

# Read
pool.with do |conn|
  result = conn.exec("SELECT 'Hello, DSQL!'")
  puts result[0]["?column?"]
end

# Write — you must wrap writes in a transaction
pool.with do |conn|
  conn.transaction do
    conn.exec_params("INSERT INTO users (id, name) VALUES (gen_random_uuid(), $1)",
 ["Alice"])
  end
end

pool.shutdown
```

Einzelne -Verbindung

Für einfache Skripte oder wenn kein Verbindungspooling benötigt wird:

```
conn = AuroraDsql::Pg.connect(host: "your-cluster.dsql.us-east-1.on.aws")
conn.exec("SELECT 1")
conn.close
```

Fortgeschrittene Nutzung

Host-Konfiguration

Der Connector unterstützt sowohl vollständige Cluster-Endpunkte (Region automatisch erkannt) als auch Cluster IDs (Region erforderlich):

```
# Full endpoint (region auto-detected)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws"
)

# Cluster ID (region required)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster-id",
  region: "us-east-1"
)
```

AWS Profile

Geben Sie ein AWS Profil für Anmeldeinformationen an:

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  profile: "production"
)
```

Format der Verbindungszeichenfolge

Der Konnektor unterstützt PostgreSQL-Verbindungszeichenfolgenformate:

```
postgres://[user@]host[:port]/[database][?param=value&...]
postgresql://[user@]host[:port]/[database][?param=value&...]
```

Unterstützte Abfrageparameter:`region`,,`profile`. `tokenDurationSecs`

```
# Full endpoint with profile
pool = AuroraDsql::Pg.create_pool(
  "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?profile=dev"
)
```

OCC-Wiederholung

Aurora DSQL verwendet Optimistic Concurrency Control (OCC). Wenn zwei Transaktionen dieselben Daten ändern, gewinnt die erste Transaktion, die einen Commit durchführt, und die zweite erhält einen OCC-Fehler.

Die OCC-Wiederholung ist optional. `occ_max_retries`Wird bei der Erstellung des Pools so eingestellt, dass automatische Wiederholungen mit exponentiellem Backoff und aktiviertem Jitter aktiviert sind: `pool.with`

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

pool.with do |conn|
  conn.transaction do
    conn.exec_params("UPDATE accounts SET balance = balance - $1 WHERE id = $2", [100,
 from_id])
    conn.exec_params("UPDATE accounts SET balance = balance + $1 WHERE id = $2", [100,
 to_id])
  end
end
```

> **Warning**
>
> `pool.with`wickelt Ihren Block NICHT automatisch in eine Transaktion ein. Für Schreiboperationen müssen Sie `conn.transaction` sich selbst anrufen. Bei einem OCC-Konflikt führt der Konnektor den gesamten Block erneut aus, sodass er nur Datenbankoperationen enthalten sollte und ein erneuter Versuch sicher sein sollte.

Um den Wiederholungsversuch bei einzelnen Aufrufen zu überspringen, übergeben Sie: `retry_occ: false`

```
pool.with(retry_occ: false) do |conn|
  conn.exec("SELECT 1")
end
```

Konfigurationsoptionen

Feld	Typ	Standard	Description
Host	Zeichenfolge	(erforderlich)	Cluster-Endpunkt oder Cluster-ID
Region	Zeichenfolge	(automatisch erkannt)	AWS Region; erforderlich, wenn es sich bei dem Host um eine Cluster-ID handelt
user	Zeichenfolge	„Administrator“	Datenbankbenutzer
Datenbank	Zeichenfolge	„Postgres“	Datenbankname
port	Ganzzahl	5432	Datenbankport
Profil	Zeichenfolge	Null	AWS Profilname für Anmeldein formationen
token_duration	Ganzzahl	900 (15 Minuten)	Gültigkeitsdauer des Tokens in Sekunden (maximal zulässig: 1 Woche, Standard: 15 Minuten)
credentia ls_provider	Aws:: Anmeldein formationen	Null	Anbieter für benutzerdefinierte Anmeldeinformationen
max_lifetime	Ganzzahl	3300 (55 Minuten)	Maximale Verbindungslebensdauer in Sekunden
applicati on_name	Zeichenfolge	Null	ORM-Präfix für application_name
Logger	Logger	Null	Logger für OCC-Wiederholungsw arnungen
occ_max_retries	Ganzzahl	nil (deaktiviert)	Max. OCC-Wiederholungsversuche aktiviertpool.with ; aktiviert Wiederholungsversuche, wenn diese Einstellung gesetzt ist

`create_pool`akzeptiert auch ein `pool:` Schlüsselwort mit einem Hash von Optionen, an die Sie direkt übergeben werden. `ConnectionPool.new` Wenn Sie es weglassen`pool:`, ist der Connector standardmäßig auf. `{size: 5, timeout: 5}` Schlüssel, die Sie angeben, haben nur Vorrang vor diesen spezifischen Standardwerten.

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  pool: { size: 10, timeout: 10 }
)
```

Authentifizierung

Der Connector verarbeitet automatisch die Aurora-DSQL-Authentifizierung, indem er mithilfe von AWS Anmeldeinformationen Token generiert. Wenn Sie die AWS Region nicht angeben, analysiert der Connector sie anhand des Hostnamens.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie unterAuthentifizierung und Autorisierung für Aurora DSQL.

Admin im Vergleich zu normalen Benutzern

- Benutzer mit dem Namen „admin“ verwenden automatisch Admin-Authentifizierungstoken
- Alle anderen Benutzer verwenden reguläre Authentifizierungstoken
- Der Connector generiert dynamisch Token für jede Verbindung

Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem .NET-Connector

Der Amazon Aurora DSQL Connector for .NET ist ein auf Npgsql basierender .NET-Connector, der die IAM-Authentifizierung für die Verbindung von.NET-Anwendungen mit Amazon Aurora DSQL-Clustern integriert.

Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und das Verbindungspooling, sodass Sie sich auf Ihre Anwendungslogik konzentrieren können.

Über den Konnektor

Amazon Aurora DSQL erfordert eine IAM-Authentifizierung mit zeitlich begrenzten Token, die vorhandene.NET-PostgreSQL-Treiber nicht nativ unterstützen. Der Aurora DSQL Connector for .NET

fügt zusätzlich zu Npgsql eine Authentifizierungsebene hinzu, die die IAM-Token-Generierung übernimmt, sodass Sie eine Verbindung zu Aurora DSQL herstellen können, ohne Ihre bestehenden Npgsql-Workflows zu ändern.

Was ist Aurora DSQL-Authentifizierung?

In Aurora DSQL umfasst die Authentifizierung:

- IAM-Authentifizierung: alle Verbindungen verwenden eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token
- Token-Generierung: Der Connector generiert Authentifizierungstoken mithilfe von AWS Anmeldeinformationen, und diese Token haben eine konfigurierbare Lebensdauer

Der Aurora DSQL Connector for .NET versteht diese Anforderungen und generiert beim Verbindungsaufbau automatisch IAM-Authentifizierungstoken.

Features

- Automatische IAM-Authentifizierung — Übernimmt die Generierung und Aktualisierung von Aurora DSQL-Tokens
- Basiert auf Npgsql — Umschließt den beliebten PostgreSQL-Treiber für.NET
- Nahtlose Integration — Funktioniert mit bestehenden Npgsql-Workflows
- Verbindungspooling — Integrierte Unterstützung `NpgsqlDataSource` mit maximaler Nutzungsdauer
- Automatische Erkennung von Regionen — Extrahiert die AWS Region aus dem Hostnamen des Aurora DSQL-Clusters
- AWS Unterstützung für Anmeldeinformationen — Unterstützt AWS Profile und Anbieter von benutzerdefinierten Anmeldeinformationen
- OCC-Wiederholung — Opt-In für optimistische Parallelitätssteuerung mit exponentiellem Backoff
- SSL-Durchsetzung — Verwendet immer SSL mit Modus und direkter TLS-Aushandlung `verify-full`

Beispiel

Ein vollständiges Beispiel finden Sie in der Beispielanwendung unter GitHub.

Schnellstart-Anleitung

Voraussetzungen

- .NET 8.0 oder höher
- Zugriff auf einen Aurora DSQL-Cluster
- AWS konfigurierte Anmeldeinformationen (über AWS CLI, Umgebungsvariablen oder IAM-Rollen)

Installation

Fügen Sie das Paket zu Ihrem Projekt hinzu:

```
dotnet add package Amazon.AuroraDsql.Npgsql
```

Usage

Verbindung zum Pool

```
using Amazon.AuroraDsql.Npgsql;

// Create a connection pool
await using var ds = await AuroraDsql.CreateDataSourceAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws",
    OccMaxRetries = 3
});

// Read
await using (var conn = await ds.OpenConnectionAsync())
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT 'Hello, DSQL!'";
    var greeting = await cmd.ExecuteScalarAsync();
    Console.WriteLine(greeting);
}

// Transactional write with OCC retry
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "INSERT INTO users (id, name) VALUES (gen_random_uuid(), @name)";
```

```
    cmd.Parameters.AddWithValue("name", "Alice");
    await cmd.ExecuteNonQueryAsync();
});
```

Einzelne -Verbindung

Für einfache Skripte oder wenn Sie kein Verbindungspooling benötigen:

```
await using var conn = await AuroraDsql.ConnectAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws"
});

await using var cmd = conn.CreateCommand("SELECT 1");
await cmd.ExecuteScalarAsync();
```

OCC-Versuch wiederholen

Aurora DSQL verwendet Optimistic Concurrency Control (OCC). Wenn zwei Transaktionen dieselben Daten ändern, gewinnt die erste Transaktion, die einen Commit durchführt, und die zweite erhält einen OCC-Fehler.

Die OCC-Wiederholung ist optional. `OccMaxRetries`In der Konfiguration so eingestellt, dass automatische Wiederholungen mit exponentiellem Backoff und Jitter aktiviert werden. Für transaktionale Schreibvorgänge verwenden: `WithTransactionRetryAsync`

```
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();

    cmd.CommandText = "UPDATE accounts SET balance = balance - 100 WHERE id = @from";
    cmd.Parameters.AddWithValue("from", fromId);
    await cmd.ExecuteNonQueryAsync();

    cmd.CommandText = "UPDATE accounts SET balance = balance + 100 WHERE id = @to";
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("to", toId);
    await cmd.ExecuteNonQueryAsync();
});
```

Verwenden Sie für DDL- oder Einzelanweisungen: `ExecWithRetryAsync`

```
await ds.ExecWithRetryAsync("CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY,
 name TEXT)");
```

> **Important**
>
> `WithTransactionRetryAsync`verwaltet`BEGIN`/`COMMIT`/`ROLLBACK`intern und öffnet für jeden Versuch eine neue Verbindung. Ihr Rückruf sollte nur Datenbankoperationen enthalten und sicher sein, dass Sie es erneut versuchen können.

Konfigurationsoptionen

Der Konnektor akzeptiert auch `postgresql://` Verbindungszeichenfolgen mit `postgres://` `region` und `profile` Abfrageparametern.

Feld	Typ	Standard	Description
`Host`	`string`	(erforderlich)	Cluster-Endpunkt oder Cluster-ID mit 26 Zeichen
`Region`	`string?`	(automatisch erkannt)	AWS Region; erforderlich, wenn `Host` es sich um eine Cluster-ID handelt
`User`	`string`	`"admin"`	Datenbankbenutzer
`Database`	`string`	`"postgres"`	Datenbankname
`Port`	`int`	`5432`	Datenbankport
`Profile`	`string?`	`null`	AWS Profilname für Anmeldein formationen
`CustomCredentialsProvider`	`AWSCredentials?`	`null`	Anbieter AWS für benutzerdefinierte Anmeldeinformationen
`TokenDurationSecs`	`int?`	`null`(SDK-Standard, 900s)	Gültigkeitsdauer des Tokens in Sekunden

Feld	Typ	Standard	Description
`OccMaxRetries`	`int?`	`null`(deaktiviert)	Standardmäßige maximale Anzahl an OCC-Wiederholungen für Wiederholungsmethoden in der Datenquelle
`OrmPrefix`	`string?`	`null`	ORM-Präfix wird vorangestellt `application_name`
`LoggerFactory`	`ILoggerFactory?`	`null`	Logger-Fabrik für Wiederholungswarnungen und Diagnosen
`ConfigureConnectionString`	`Action<NpgsqlConnectionStringBuilder>?`	`null`	Callback, um Pool-Einstellungen zu überschreiben oder zusätzliche Eigenschaften der Npgsql-Verbindungszeichenfolge festzulegen. SSL und `Enlist` sind Sicherheitsinvarianten und können nicht außer Kraft gesetzt werden.

Authentifizierung

Der Connector verarbeitet automatisch die Aurora-DSQL-Authentifizierung, indem er mithilfe von AWS Anmeldeinformationen Token generiert. Wenn Sie die AWS Region nicht angeben, analysiert der Connector sie anhand des Hostnamens.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie unterAuthentifizierung und Autorisierung für Aurora DSQL.

Admin im Vergleich zu normalen Benutzern

- Benutzer mit dem Namen „admin“ verwenden automatisch Admin-Authentifizierungstoken
- Alle anderen Benutzer verwenden reguläre Authentifizierungstoken
- Der Connector generiert dynamisch Token für jede Verbindung

Zugreifen auf Aurora DSQL mit PostgreSQL-kompatiblen Clients

Aurora DSQL verwendet das PostgreSQL-Wire-Protokoll. Sie können eine Verbindung zu PostgreSQL mithilfe einer Vielzahl von Tools und Clients herstellen, z. B. psql AWS CloudShell, DBeaver und. DataGrip Die folgende Tabelle fasst zusammen, wie Aurora DSQL allgemeine PostgreSQL-Verbindungsparameter zuordnet:

PostgreSQL	Aurora DSQL	Hinweise
Rolle (auch Benutzer oder Gruppe genannt)	Datenbankrolle	Aurora DSQL erstellt eine Rolle für Sie mit dem Namen `admin`. Wenn Sie benutzerd efinierte Datenbankrollen erstellen, müssen Sie die `admin`-Rolle verwenden, um sie IAM-Rollen für die Authentifizierung zuzuordnen, wenn Sie eine Verbindung zu Ihrem Cluster herstellen. Weitere Informationen finden Sie unter Verwenden von Datenbankrollen und IAM-Authentifizierung.
Host (auch hostname oder hostspec genannt)	Cluster-Endpunkt	Aurora DSQL-Einzelregions-Cluster bieten einen einzigen verwalteten Endpunkt und leiten den Datenverkehr automatisch um, falls es innerhalb der Region zu einer Nichtverf ügbarkeit kommt.
Port	Nicht zutreffend – Standard-`5432` verwenden	Dies ist der PostgreSQL-Standard.
Datenbank (dbname)	`postgres` verwenden	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 SSL-Modus `require`. Verbindungen ohne SSL werden von Aurora DSQL abgelehnt.
Passwort	Authentifizierungs token	Aurora DSQL erfordert temporäre Authentif izierungstokens anstelle von dauerhaft

PostgreSQL	Aurora DSQL	Hinweise
		gültigen Passwörtern. Weitere Informationen hierzu finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

Beim Herstellen einer Verbindung benötigt Aurora DSQL anstelle eines herkömmlichen Passworts ein signiertes IAM-Authentifizierungstoken. Diese temporären Token werden mit AWS Signature Version 4 generiert und nur beim Verbindungsaufbau verwendet. Sobald die Verbindung hergestellt ist, bleibt die Sitzung aktiv, bis sie endet oder der Client die Verbindung trennt.

Wenn Sie versuchen, eine neue Sitzung mit einem abgelaufenen Token zu öffnen, schlägt die Verbindungsanforderung fehl und es muss ein neues Token generiert werden. Weitere Informationen finden Sie unter Generieren eines Authentifizierungstokens in Amazon Aurora DSQL.

Greifen Sie mithilfe von SQL-Clients auf Aurora DSQL zu

Aurora DSQL unterstützt mehrere PostgreSQL-kompatible Clients für die Verbindung zu Ihrem Cluster. In den folgenden Abschnitten wird beschrieben, wie Sie mithilfe von PostgreSQL mit AWS CloudShell oder Ihrer lokalen Befehlszeile sowie GUI-basierte Tools wie und eine Verbindung herstellen. DBeaver JetBrains DataGrip Jeder Client benötigt ein gültiges Authentifizierungstoken, wie im vorherigen Abschnitt beschrieben.

Topics

- Für den DBeaver Zugriff auf Aurora DSQL verwenden
- Für den JetBrains DataGrip Zugriff auf Aurora DSQL verwenden
- Verwenden Sie das interaktive PostgreSQL-Terminal (psql), um auf Aurora DSQL zuzugreifen
- Verwenden Sie den Aurora DSQL-Treiber für SQLTools
- Fehlerbehebung

Für den DBeaver Zugriff auf Aurora DSQL verwenden

DBeaver ist ein universeller SQL-Client, mit dem jede Datenbank verwaltet werden kann, die über einen JDBC-Treiber verfügt. Aufgrund seiner robusten Funktionen zum Anzeigen, Bearbeiten und Verwalten von Daten wird er häufig von Entwicklern und Datenbankadministratoren verwendet.

Mithilfe DBeaver der Cloud-Konnektivitätsoptionen können Sie sich nativ mit Aurora DSQL verbinden DBeaver .

DBeaver Profi

DBeaver PRO-Produkte bieten ab Version 25.3 eine native Integration mit Aurora DSQL. Folgen Sie den Anweisungen in der DBeaver Dokumentation, um eine Verbindung zu Ihrem Aurora DSQL-Cluster herzustellen.

DBeaver Community-Ausgabe

DBeaver Die Community Edition ist die kostenlose Open-Source-Version. Installationsanweisungen finden Sie auf der Download-Seite. Um von der DBeaver Community Edition aus eine Verbindung zu DSQL herzustellen, müssen Sie das Aurora DSQL-Plugin für installieren. DBeaver

Das Aurora DSQL Plugin für DBeaver baut auf dem Aurora DSQL Connector für JDBC auf und ermöglicht die IAM-Authentifizierung für Aurora DSQL-Cluster. Es lässt sich bequem über die DBeaver Benutzeroberfläche installieren und macht das Schreiben von Code zur Token-Generierung oder die manuelle Bereitstellung eines gültigen IAM-Tokens überflüssig. Dadurch wird die Authentifizierung vereinfacht und gleichzeitig Sicherheitsrisiken vermieden, die mit herkömmlichen benutzergenerierten Passwörtern verbunden sind.

Features

- Support der IAM-Authentifizierung: Stellen Sie mithilfe von AWS-IAM-Anmeldeinformationen eine Verbindung zu Aurora DSQL-Clustern her, um eine sichere, kennwortfreie Authentifizierung zu gewährleisten
- Automatisches Treibermanagement: Reibungslose Installation und Konfiguration des Aurora DSQL Connector für JDBC
- Flexible Verbindungsoptionen: Wählen Sie zwischen einer hostbasierten oder einer JDBC-URL-basierten Verbindungskonfiguration

Aurora DSQL-Plugin für die Installation DBeaver

1. Wenn es DBeaver geöffnet ist, gehen Sie zum Drop-down-Menü Hilfe → Neue Software installieren
2. Klicken Sie auf Hinzufügen, um ein neues Repository hinzuzufügen
3. Geben Sie ein:
   - Name (Name: `Aurora DSQL Plugin`

- Ort: `https://awslabs.github.io/aurora-dsql-dbeaver-plugin/update-site/`

4. Überprüfen Sie den Aurora DSQL Connector für JDBC
5. Klicken Sie auf Weiter, akzeptieren Sie die Lizenz und schließen Sie die Installation ab
6. DBeaver Wenn Sie dazu aufgefordert werden, starten

Eine Aurora DSQL-Verbindung erstellen

1. Klicken Sie auf Neue Datenbankverbindung
2. Wählen Sie Aurora DSQL
3. Wählen Sie unter Server eine der folgenden Optionen für die Einstellung Connect durch
   - Host
     - um die Texteingaben der Benutzeroberfläche für die folgenden Felder zu aktivieren:
       - Endpunkt: DSQL-Cluster-Endpunkt
       - Benutzername: DSQL-Benutzername (z. B. admin)
       - AWS-Profil: z. B. default — Das Standardprofil, das verwendet wird, wenn kein bestimmtes Profil angegeben ist
       - AWS-Region (optional): muss mit der Region übereinstimmen, in der Ihr DSQL-Cluster existiert, andernfalls schlägt die Authentifizierung fehl
   - URL
     - JDBC-URL in diesem Format:

       ```
       jdbc:aws-dsql:postgresql://{cluster_endpoint}/{database}?
       user=admin&profile=default&region=us-east-1
       ```

     - Hinweis: In diesem Modus ist nur die URL-Eingabe aktiviert. Um der JDBC-Verbindungszeichenfolge Parameter hinzuzufügen, verwenden Sie das URL-Abfrageparameterformat, das mit? beginnt als ersten Parameter und fügen Sie ein & für nachfolgende Parameter an.
4. Klicken Sie auf Verbindung testen, um zu überprüfen, ob die Aurora DSQL-Verbindung funktioniert
5. Klicken Sie auf Fertigstellen

Fehlerbehebung

Problem mit dem Windows Trust Store

Windows-Benutzer können Probleme beim Herunterladen des Aurora DSQL Connector for JDBC-Treibers von Maven Central haben.

Ursache: Der Windows Trust Store enthält möglicherweise nicht die Zertifikate, die für den Zugriff auf das Maven Central-Repository erforderlich sind.

Lösung:

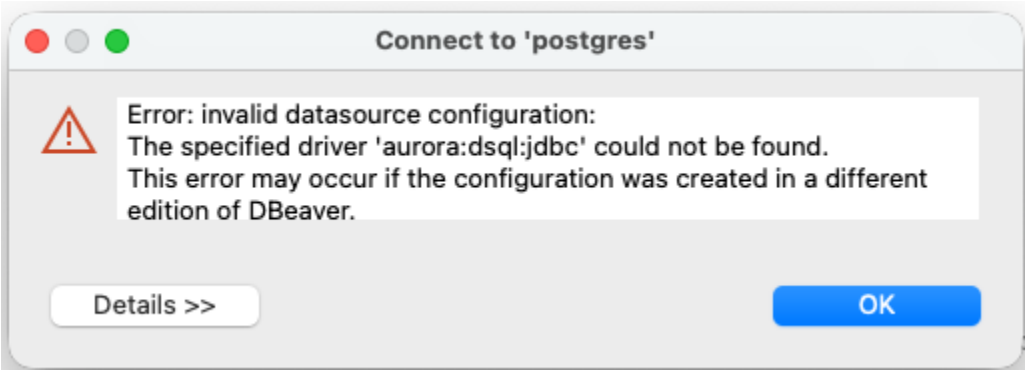
1. DBeaver Als „Administrator“ ausführen
2. Deaktivieren Sie diese Einstellung — Windows > Einstellungen > Verbindungen > „Windows Trust Store verwenden“

Fehler beim fehlenden Treiber

Wenn Sie ein fehlendes Treibersymbol oder Verbindungsfehler sehen, ist das Aurora DSQL (Community Plugin) möglicherweise nicht in Ihrer aktuellen DBeaver Version installiert. Nachfolgend finden Sie einige Beispiele für Fehler und deren Behebung:

- Eine neue Verbindung mit dem fehlenden Treiber erstellen:

- Es wird versucht, eine Verbindung ohne den Treiber herzustellen:



Ursache: Wenn mehrere DBeaver Versionen installiert sind, werden die Verbindungseinstellungen gemeinsam genutzt, aber die Treiber werden pro Anwendung installiert.

Lösung: Installieren Sie das Aurora DSQL (Community-Plugin) erneut, indem Sie die obigen Installationsschritte befolgen.

> **Important**
>
> Die von DBeaver for PostgreSQL-Datenbanken bereitgestellten Verwaltungsfunktionen (wie Session Manager und Lock Manager) gelten aufgrund ihrer einzigartigen Architektur nicht für Aurora DSQL-Datenbanken. Diese Bildschirme sind zwar zugänglich, bieten jedoch keine zuverlässigen Informationen über den Zustand oder den Status der Datenbank.

Für den JetBrains DataGrip Zugriff auf Aurora DSQL verwenden

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 JetBrainsWebsite.

Um eine neue Aurora DSQL-Verbindung einzurichten in JetBrains DataGrip

1. Klicken Sie auf Neue Datenquelle und wählen Sie dann PostgreSQL aus.
2. Geben Sie auf der Registerkarte Datenquellen/Allgemein die folgenden Informationen ein:
   - Host – Verwenden Sie Ihren Cluster-Endpunkt.

     Port – Aurora DSQL verwendet den PostgreSQL-Standardwert: `5432`

     Datenbank – Aurora DSQL verwendet den PostgreSQL-Standardwert `postgres`

     Authentifizierung – Wählen Sie `User & Password` aus.

     Benutzername – Geben Sie ein `admin`

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

     URL – Dieses Feld muss unverändert bleiben. 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 festlegen. Aurora DSQL unterstützt `PGSSLMODE=require and PGSSLMODE=verify-full`. Aurora DSQL erzwingt serverseitig die SSL-Kommunikation und lehnt Verbindungen ohne SSL ab. Für `verify-full` diese Option müssen Sie die SSL-Zertifikate lokal installieren. Weitere Informationen finden Sie unter SSL/TLS-Zertifikate.

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

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

Verwenden Sie das interaktive PostgreSQL-Terminal (psql), um auf Aurora DSQL zuzugreifen

Wird verwendet AWS CloudShell , um mit dem interaktiven PostgreSQL-Terminal (psql) auf Aurora DSQL zuzugreifen

Gehen Sie wie folgt vor, um mit dem interaktiven PostgreSQL-Terminal von auf Aurora DSQL zuzugreifen. AWS CloudShellWeitere Informationen finden Sie unter Was ist. AWS CloudShell

Um eine Verbindung herzustellen mit AWS CloudShell

1. Melden Sie sich bei der Aurora DSQL-Konsole an.
2. Wählen Sie den Cluster aus, für den Sie sich öffnen möchten. CloudShell Wenn Sie noch keinen Cluster erstellt haben, folgen Sie den Schritten unter Schritt 1: Erstellen eines Einzelregions-Clusters mit Aurora DSQL oder Erstellen eines Clusters mit mehreren Regionen.
3. Wählen Sie Mit dem Abfrage-Editor Connect und dann Connect mit CloudShell.

4. Wählen Sie aus, ob Sie eine Verbindung als Administrator oder mit einer benutzerdefinierten Datenbankrolle herstellen möchten.
5. Wählen Sie Launch in CloudShell und wählen Sie im folgenden CloudShell Dialog Run.

Verwenden Sie die lokale CLI, um mit dem interaktiven PostgreSQL-Terminal (psql) auf Aurora DSQL zuzugreifen

Verwenden Sie`psql`, ein terminalbasiertes Front-End-Hilfsprogramm für PostgreSQL, um interaktiv Abfragen einzugeben, sie an PostgreSQL weiterzugeben und die Abfrageergebnisse anzuzeigen.

> **Note**
>
> Verwenden Sie den PostgreSQL-Client der Version 17, um die Antwortzeiten für Abfragen zu verbessern. Wenn Sie die CLI in einer anderen Umgebung verwenden, stellen Sie sicher, dass Sie Python Version 3.8+ und psql Version 14+ manuell einrichten.

Laden Sie das Installationsprogramm Ihres Betriebssystems von der PostgreSQL-Downloadseite herunter. Weitere Informationen `psql` finden Sie unter PostgreSQL Client Applications auf der PostgreSQL-Website.

Wenn Sie den bereits AWS CLI installiert haben, verwenden Sie das folgende Beispiel, um eine Verbindung zu Ihrem Cluster herzustellen.

```
# 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
```

Verwenden Sie den Aurora DSQL-Treiber für SQLTools

Der Aurora DSQL-Treiber für SQLTools ist eine Visual Studio Code-Erweiterung für Amazon Aurora DSQL, die sich in integrieren lässt. SQLTools Es ermöglicht Entwicklern, direkt von VS Code aus eine Verbindung zu Aurora DSQL-Datenbanken herzustellen und diese abzufragen. Der Treiber kann über Visual Studio Marketplace und Open VSX Registry installiert werden. Kiro, Cursor und andere Anbieter VSCode IDEs können die Open VSX Registry verwenden, um den Treiber gemäß dem auf dieser Seite beschriebenen Standardinstallationsverfahren zu installieren.

Features

- Automatische IAM-Authentifizierung
- Standarddatenbankoperationen wie das Durchsuchen von Schemas und Tabellen und das Ausführen von SQL-Abfragen.

Installation

1. Öffnen Sie die Ansicht „Erweiterungen“.
2. Suchen Sie nach „Aurora DSQL-Treiber für SQLTools“.
3. Klicken Sie auf „Installieren“.

Hinweis:

Die SQLToolsErweiterung wird automatisch installiert, sofern sie nicht bereits vorhanden ist.

Authentifizierung

In Aurora DSQL verwenden alle Verbindungen eine IAM-basierte Authentifizierung mit zeitlich begrenzten Token. Der Treiber verarbeitet automatisch die Aurora-DSQL-Authentifizierung mithilfe des Aurora DSQL Connector für Node-Postgres.

Weitere Informationen zur Authentifizierung in Aurora DSQL finden Sie im Benutzerhandbuch.

Eine Aurora DSQL-Verbindung erstellen

Voraussetzungen

- AWS-Anmeldeinformationen konfiguriert (über AWS-CLI, Umgebungsvariablen oder IAM-Rollen)

Schritte

1. Klicken Sie auf das SQLTools Symbol in der linken Seitenleiste.
2. Zeigen SQLTools Sie im Bereich mit der Maus auf VERBINDUNGEN und klicken Sie auf das Symbol Neue Verbindung hinzufügen.
3. Wählen Sie auf der Registerkarte SQLTools Einstellungen den Aurora DSQL-Treiber aus der Liste aus.
4. Geben Sie die Verbindungsparameter ein.
   - AWS-Region
     - Optional — Die Region wird vom Aurora DSQL-Cluster-Endpunkt aus analysiert.
     - Erforderlich, wenn im Feld DSQL-Cluster nur eine Cluster-ID angegeben ist.
   - AWS-Profil
     - Wird für die Token-Generierung verwendet.
     - Verwendet das Standardprofil, falls nicht angegeben.
5. Klicken Sie auf die Schaltfläche „Verbindung testen“, um die Verbindung zu testen.
6. Klicken Sie auf Verbindung speichern.

Fehlerbehebung

Ablauf der Authentifizierungsdaten für die SQL-Clients

Bestehende Sitzungen bleiben für maximal 1 Stunde authentifiziert oder bis eine explizite Trennung oder ein clientseitiger Timeout erfolgt. Wenn neue Verbindungen hergestellt werden müssen, muss ein neues Authentifizierungstoken generiert und im Feld Passwort der Verbindung bereitgestellt werden. Der Versuch, eine neue Sitzung zu öffnen (z. B. um neue Tabellen aufzulisten oder eine neue SQL-Konsole zu öffnen), erzwingt einen neuen Authentifizierungsversuch. Ist das in den Verbindungseinstellungen konfigurierte Token nicht mehr gültig, schlägt die neue Sitzung fehl und alle zuvor geöffneten Sitzungen werden ungültig. Denken Sie daran, wenn Sie die Dauer Ihres IAM-Authentifizierungstokens mit der `expires-in` Option wählen, die standardmäßig auf 15 Minuten und auf einen Höchstwert von sieben Tagen festgelegt werden kann.

Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Aurora DSQL-Dokumentation.

Konnektivitätstools für Amazon Aurora DSQL-Cluster

Aurora DSQL ist mit vielen Datenbanktreibern und ORM-Bibliotheken von Drittanbietern kompatibel. AWS bietet zwei Arten von Tools, um die Arbeit mit Aurora DSQL zu vereinfachen:

- Konnektoren — Authentifizierungs-Plugins, die Datenbanktreiber so erweitern, dass sie die automatische Generierung von IAM-Tokens übernehmen. Verwenden Sie Konnektoren, wenn Sie direkt mit Datenbanktreibern arbeiten.
- Adapter und Dialekte — Erweiterungen für bestimmte ORM-Frameworks, die IAM-Authentifizierung und verbesserte Aurora DSQL-Kompatibilität bieten. Verwenden Sie Adapter, wenn Sie mit einem unterstützten ORM-Framework arbeiten.

Aurora DSQL-Adapter und -Dialekte

Die folgende Tabelle zeigt die verfügbaren Adapter und Dialekte für Aurora DSQL.

Programmiersprache	ORM/Framework	Link zum Repository
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy
Python	Tortoise ORM	https://github.com/awslabs/aurora-dsql-orms/-orm tree/main/python/tortoise

Beispiele für Datenbanktreiber

Die folgende Tabelle zeigt Beispielcode für die Verbindung mit Aurora DSQL mithilfe von Datenbanktreibern von Drittanbietern.

Programmiersprache	Treiber	Repository-Linkbeispiel
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	HikariCP + PGJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	Node-Postgres (AWS Lambda)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/lambda
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

Programmiersprache	Treiber	Repository-Linkbeispiel
Python	asyncpg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/asyncpg
Python	Psychokopy	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycho P 2	https://github.com/aws-samples/aurora-dsql-samples/2tree/main/python/psycopg
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx

ORM- und Framework-Beispiele

Die folgende Tabelle zeigt Beispielcode für die Verwendung von ORM-Bibliotheken und Frameworks von Drittanbietern mit Aurora DSQL.

Programmiersprache	ORM/Framework	Repository-Linkbeispiel
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate/examples/pet-Klinik-App
Java	Liquibase	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/liquibase

Programmiersprache	ORM/Framework	Repository-Linkbeispiel
Java	Spring Boot	https://github.com/aws-samples/aurora-dsql-samples/_boottree/main/java/spring
Python	Django	https://github.com/awslabs/aurora-dsql-orms/-Klinik-Apptree/main/python/django/examples/pet
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy/examples/pet-Klinik-App
Python	Schildkröte ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm/example
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails
TypeScript	Prisma	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/prisma
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/-ormtree/main/typescript/type

Daten in Aurora DSQL laden

Ganz gleich, ob Sie aus einer vorhandenen Datenbank migrieren, Dateien aus Amazon Simple Storage Service importieren oder Daten aus Ihrem lokalen System laden, Aurora DSQL bietet mehrere Möglichkeiten, Ihre Daten zu importieren. In diesem Abschnitt werden die empfohlenen Tools und Techniken für Datenladungen aller Größen, von Gigabyte bis hin zu Hunderten von Terabyte, behandelt.

Wählen Sie einen Ladeansatz

Aurora DSQL unterstützt standardmäßige PostgreSQL-Befehle zum Laden von Daten, aber das effiziente Laden von Daten in großem Maßstab erfordert Parallelisierung, Verbindungsmanagement und Fehlerbehebung. In der folgenden Tabelle sind Ihre Optionen zusammengefasst:

Ansatz	Am besten geeignet für	Überlegungen
Aurora DSQL Loader - Open-Source-Hilfsprogramm, das das Parallelisieren von Einfügungen bei Verwendung von Aurora DSQL erleichtert	Die meisten Szenarien zum Laden von Daten, insbesondere Migrationen und Massenimporte	Verwaltet Parallelisierung, Verbindungspooling, Konfliktlösung und IAM-Authentifizierung automatisch. Als Quellcode oder Binärcode verfügbar.
PostgreSQL **`\copy`** - Clientseitiger Metabefehl `psql`	Wird einfach geladen, wenn Sie bereits verbunden sind über `psql`	Liest Dateien auf dem Client und streamt Daten über die Verbindung; Sie verwalten die Parallelisierung selbst
`INSERT`Transaktionen — Standard `SQL DML`	Kleine Datensätze oder anwendungsorientierte Beilagen	Einfachster Ansatz, aber am langsamsten für Massendaten

Verwenden Sie für die meisten Aufgaben zum Laden von Daten den Aurora DSQL Loader. Es bewältigt die betriebliche Komplexität beim Laden von Daten in eine verteilte Datenbank, einschließlich der parallel Ausführung über mehrere Verbindungen und der automatischen Wiederholung fehlgeschlagener Operationen.

Aurora SQL-Lader

Der Aurora DSQL Loader ist ein Open-Source-Befehlszeilenprogramm, das entwickelt wurde, um Daten effizient in Aurora DSQL-Cluster zu laden. Es verwaltet das Verbindungspooling, parallelisiert die Datenübertragung zwischen mehreren Workern und behandelt Konflikte und Wiederholungsversuche automatisch.

Schlüssel-Features

Der Aurora DSQL Loader bietet die folgenden Funktionen:

Paralleles Laden

: Konfigurierbare Worker-Threads ermöglichen das gleichzeitige Laden von Daten über mehrere Verbindungen und verbessern so die Leistung.

Verbindungspooling

: Verwaltet einen Pool von Verbindungen zu Ihrem Aurora DSQL-Cluster und wickelt die IAM-Authentifizierung und den Verbindungslebenszyklus automatisch ab.

Unterstützung mehrerer Dateiformate

: Unterstützt CSV (kommagetrennte Werte), TSV (tabulatorgetrennte Werte) und das Apache Parquet-Spaltenformat. Der Loader erkennt das Dateiformat automatisch anhand der Quell-URI-Erweiterung.

Automatische Schemainferenz

: Bei Verwendung mit dem `--if-not-exists` Flag kann der Loader auf der Grundlage der Daten automatisch Tabellen mit entsprechenden Spaltentypen erstellen.

Umgang mit Konflikten

: Wenn Ihre Zieltabelle eindeutige Einschränkungen hat, konfigurieren Sie, wie der Loader Konflikte behandelt, indem Sie die folgende `--on-conflict` Option verwenden: Duplikate überspringen, Datensätze verschieben oder einen Fehler zurückgeben.

Fehlertoleranz

Automatische Wiederholungsversuche und Funktionen zur Wiederaufnahme von Aufträgen stellen sicher, dass unterbrochene Ladevorgänge von ihrem Endpunkt aus fortgesetzt werden können, anstatt sie vollständig neu zu starten.

Lokale Quellen und S3-Quellen

Laden Sie mit S3 Daten aus lokalen Dateisystempfaden oder direkt aus Amazon S3-Buckets. URIs

Voraussetzungen

Bevor Sie den Aurora DSQL Loader verwenden, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Ein aktiver Aurora DSQL-Cluster mit einem gültigen Endpunkt.
- AWS Anmeldeinformationen, die über die Rollen AWS CLI (aws configure), AWS Single Sign-On (aws sso login) oder IAM konfiguriert wurden.
- IAM-Berechtigungen: `dsql:DbConnectAdmin` oder `dsql:DbConnect` auf Ihrem Aurora DSQL-Cluster.
- Für S3-Quellen entsprechende Berechtigungen zum Lesen aus dem Quell-Bucket.

Installation

Laden Sie die neueste Version von der GitHub Releases-Seite herunter. Vorgefertigte Binärdateien sind für gängige Plattformen verfügbar. Anweisungen zum Erstellen aus dem Quellcode finden Sie im Aurora DSQL Loader-Repository.

Verwendungsbeispiele

Die folgenden Beispiele zeigen allgemeine Anwendungsfälle für den Aurora DSQL Loader.

Example Eine lokale CSV-Datei wird geladen

In diesem Beispiel wird eine CSV-Datei aus Ihrem lokalen Dateisystem in eine bestehende Tabelle geladen:

```
aurora-dsql-loader load \
```

```
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table
```

Example Daten aus Amazon S3 laden

In diesem Beispiel wird eine Parquet-Datei aus einem Amazon S3 S3-Bucket geladen:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/data.parquet \
  --table my_table
```

Example Automatische Tabellenerstellung

In diesem Beispiel wird automatisch eine neue Tabelle auf der Grundlage des Datenschemas erstellt:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --if-not-exists
```

Example Vor dem Laden validieren

In diesem Beispiel wird Ihre Konfiguration validiert, ohne tatsächlich Daten zu laden:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --dry-run
```

Example Wiederaufnahme eines unterbrochenen Ladevorgangs

Wenn ein Ladevorgang unterbrochen wird, können Sie ihn mit der Job-ID aus dem vorherigen Lauf fortsetzen:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
```

```
  --table my_table \
  --resume-job-id job-id \
  --manifest-dir ./loader-state
```

> **Note**
>
> Bei der Wiederaufnahme überspringt der Loader die meisten bereits abgeschlossenen Arbeiten, versucht es jedoch möglicherweise erneut mit einigen Datensätzen. Wenn Ihre Zieltabelle eindeutige Einschränkungen hat, verwenden Sie die `--on-conflict` Option, um Duplikate zu behandeln, z. B. um sie zu überspringen oder zu aktualisieren. `DO NOTHING` `DO UPDATE`

Befehlszeilenoptionen

Der Aurora DSQL Loader unterstützt die folgenden Befehlszeilenoptionen:

`--endpoint`

(Erforderlich) Der Aurora DSQL-Cluster-Endpunkt. Beispiel: *`cluster-id`*`.dsql.`*`region`*`.on.aws`

`--source-uri`

(Erforderlich) Der Pfad zur Datendatei. Kann ein lokaler Dateipfad oder ein S3-URI sein (z. B.`s3://`*`bucket-name`*`/`*`file.parquet`*).

`--table`

(Erforderlich) Der Name der Zieltabelle in Ihrer Aurora DSQL-Datenbank.

`--if-not-exists`

(Optional) Erstellen Sie automatisch die Zieltabelle, falls sie nicht existiert. Der Loader leitet das Schema aus den Daten ab.

`--dry-run`

(Optional) Überprüfen Sie die Konfiguration und die Daten, ohne sie tatsächlich in die Datenbank zu laden.

`--resume-job-id`

(Optional) Setzt einen zuvor unterbrochenen Ladevorgang mit der angegebenen Job-ID fort.

`--manifest-dir`

(Optional) Verzeichnis zum Speichern des Jobstatus und der Manifeste, das für die Wiederaufnahme des Jobs verwendet wird.

`--on-conflict`

(Optional) Gibt an, wie Konflikte beim Einfügen von Zeilen behandelt werden sollen, die gegen eindeutige Einschränkungen in der Zieltabelle verstoßen. Gültige Werte sind `error` (einen Fehler zurückgeben), `do-nothing` (doppelte Zeilen überspringen) oder `do-update` (vorhandene Zeilen mit neuen Werten aktualisieren).

Führen Sie folgenden Befehl aus, um eine vollständige Liste der Optionen und zusätzlichen Konfigurationsparameter zu erhalten:

```
aurora-dsql-loader load --help
```

Best Practices

- Verwenden Sie den Testlauf zur Validierung — Testen Sie Ihre Ladekonfiguration immer mit, `--dry-run` bevor Sie Daten in Produktionstabellen laden.
- Definieren Sie eindeutige Einschränkungen für die Wiederaufnahme — Wenn Sie unterbrochene Ladevorgänge wieder aufnehmen müssen, definieren Sie eindeutige Einschränkungen für Ihre Zieltabellen und verwenden Sie die `--on-conflict` Option, bereits geladene Datensätze zu verarbeiten.
- Verwenden Sie Parquet für große Datensätze — Das spaltenförmige Format von Parquet bietet im Vergleich zu CSV oder TSV in der Regel eine bessere Komprimierung und ein schnelleres Laden großer Datensätze.
- Manifestverzeichnisse beibehalten — Behalten Sie das Manifestverzeichnis für Ladeaufträge bei, bis Sie bestätigen, dass der Ladevorgang erfolgreich abgeschlossen wurde, und ermöglichen Sie bei Bedarf die Wiederaufnahme.
- Tabellen nach Möglichkeit vorab erstellen — Definieren Sie die Zieltabelle mit expliziten Spaltendatentypen und Primärschlüsseln, bevor Sie Daten laden. Vorab erstellte Schemas geben Ihnen die Kontrolle über die Typgenauigkeit und Indizierung, was in der Regel zu einer besseren Abfrageleistung im Vergleich zu automatisch abgeleiteten Schemas führt.

Fehlerbehebung

Authentifizierungsfehler

Vergewissern Sie sich, dass Ihre AWS Anmeldeinformationen korrekt konfiguriert sind und dass Ihre IAM-Identität über die erforderlichen `dsql:DbConnect` OR-Berechtigungen für den Zielcluster verfügt. `dsql:DbConnectAdmin`

S3-Zugriffsfehler

Stellen Sie sicher, dass Ihre IAM-Identität über die entsprechenden S3-Leseberechtigungen für den Quell-Bucket und die Objekte verfügt.

Fehler bei der Schemainferenz

Stellen Sie bei der Verwendung sicher`--if-not-exists`, dass Ihre Datendatei konsistente Spaltentypen hat. Gemischte Typen in einer Spalte können dazu führen, dass die Schemainferenz fehlschlägt.

Doppelte Schlüsselfehler bei der Wiederaufnahme

Wenn Sie beim Wiederaufnehmen eines Ladevorgangs auf Fehler mit doppelten Schlüsseln stoßen, fügen Sie Ihrer Zieltabelle eindeutige Einschränkungen hinzu, damit der Loader bereits geladene `ON CONFLICT DO NOTHING` Datensätze überspringen kann.

Weitere Informationen zur Fehlerbehebung finden Sie im Aurora DSQL GitHub Loader-Repository.

Migrationspfade

In den folgenden Abschnitten wird beschrieben, wie Daten aus gängigen Quellsystemen nach Aurora DSQL migriert werden.

Migration von PostgreSQL

Um Daten aus einer vorhandenen PostgreSQL-Datenbank zu Aurora DSQL zu migrieren:

1. Exportieren Sie Ihre Daten aus PostgreSQL in das CSV- oder Parquet-Format. Sie können den `COPY` PostgreSQL-Befehl verwenden, um jede Tabelle zu exportieren:

```
COPY my_table TO '/path/to/my_table.csv' WITH (FORMAT csv, HEADER true);
```

2. Erstellen Sie die Zieltabelle in Aurora DSQL. Sie können das Schema entweder manuell erstellen oder das `--if-not-exists` Loader-Flag verwenden, um das Schema aus Ihren Daten abzuleiten.
3. Laden Sie die exportierten Daten mit dem Aurora DSQL Loader:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Tip**
>
> Bei großen Migrationen sollten Sie den Export in das Parquet-Format in Betracht ziehen, um eine bessere Komprimierung und ein schnelleres Laden zu erzielen. Tools wie DuckDB können CSV-Dateien effizient in Parquet konvertieren.

Migration von MySQL

Um Daten von MySQL zu Aurora DSQL zu migrieren:

1. Exportieren Sie Ihre Daten aus MySQL in das CSV-Format mit `SELECT INTO OUTFILE` oder einem ähnlichen Tool mysqldump mit der `--tab` Option:

```
SELECT * FROM my_table
INTO OUTFILE '/path/to/my_table.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Erstellen Sie die Zieltabelle in Aurora DSQL mit entsprechenden PostgreSQL-kompatiblen Datentypen.
3. Laden Sie die exportierten Daten mit dem Aurora DSQL Loader:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Note**
>
> MySQL und PostgreSQL haben unterschiedliche Datentypsysteme. Überprüfen Sie Ihr Schema und passen Sie die Datentypen nach Bedarf an, wenn Sie Tabellen in Aurora DSQL erstellen.

Aus Amazon S3 laden

Wenn sich Ihre Daten bereits in Amazon S3 befinden, können Sie sie direkt laden, ohne sie auf Ihr lokales System herunterzuladen. Der Aurora DSQL Loader unterstützt S3 URIs nativ:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/path/to/data.parquet \
  --table my_table
```

Stellen Sie sicher, dass Ihre IAM-Identität über `s3:GetObject` Berechtigungen für die Quellobjekte verfügt.

PostgreSQL\ copy verwenden

Wenn Sie bereits über eine `psql` Sitzung, die die IAM-Authentifizierung abwickelt, mit Aurora DSQL verbunden sind, können Sie den clientseitigen `\copy` Metabefehl verwenden, um Daten aus Ihrem lokalen Dateisystem zu laden. `\copy`Liest im Gegensatz zur serverseitigen `COPY` Anweisung die Datei auf dem Client-Computer und streamt die Daten über die bestehende Verbindung, sodass kein serverseitiger Dateizugriff erforderlich ist. Dieser Ansatz eignet sich gut für einfache Ladevorgänge mit nur einem Thread.

Example Laden einer CSV-Datei mit\ copy

```
\copy my_table FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

Bei `\copy` direkter Nutzung sind Sie verantwortlich für:

- Verwaltung der Parallelisierung beim Laden mehrerer Dateien oder großer Datensätze
- Verwaltung der Verbindungsverwaltung und Aktualisierung des Authentifizierungstokens
- Implementierung der Wiederholungslogik für fehlgeschlagene Operationen

Bewährte Methoden für INSERT-Transaktionen

Wenn Sie `INSERT` Anweisungen verwenden, um Daten in Aurora DSQL zu laden, gehen Sie wie folgt vor, um den Durchsatz und die Zuverlässigkeit zu verbessern:

- Batch zu mehreren Zeilen zusammenfassen INSERTs — Gruppieren Sie mehrere Zeilen zu einer einzigen `INSERT` Anweisung, um Roundtrips zu reduzieren. `INSERT INTO my_table VALUES (1, 'a'), (2, 'b'), (3, 'c')`Ist beispielsweise effizienter als drei separate Anweisungen.
- Verwenden Sie parametrisierte Abfragen — Verwenden Sie vorbereitete Anweisungen mit Parameterbindung anstelle einer Zeichenkettenverkettung. Dadurch werden SQL-Injection-Risiken vermieden und die Datenbank kann Abfragepläne wiederverwenden.
- Halten Sie die Transaktionen klein — Aurora DSQL verwendet eine optimistische Parallelitätssteuerung, sodass bei großen Transaktionen, die viele Zeilen berühren, die Wahrscheinlichkeit von Konflikten steigt. Streben Sie Transaktionen mit einigen hundert Zeilen an, anstatt mit Tausenden.
- Implementieren Sie Wiederholungslogik — In einem verteilten System sind vorübergehende Fehler wie Konflikte mit optimistischer Parallelitätskontrolle (OCC) zu erwarten. Implementieren Sie einen exponentiellen Backoff mit Wiederholungsversuchen für fehlgeschlagene Transaktionen.
- Verbindungsübergreifende Parallelisierung — Öffnen Sie mehrere Verbindungen und verteilen Sie Inserts auf diese. Jede Verbindung kann eine andere Teilmenge von Daten gleichzeitig verarbeiten.

Für die meisten Anwendungsfälle bietet der Aurora DSQL Loader einen einfacheren und robusteren Ansatz für das Laden von Daten.

Weitere Ressourcen

- Aurora DSQL Loader aktiviert GitHub — Quellcode, Dokumentation und Problemverfolgung
- Generieren eines Authentifizierungstokens in Amazon Aurora DSQL— Erfahren Sie mehr über IAM-Authentifizierungstoken für Aurora DSQL
- Zugreifen auf Aurora DSQL mit PostgreSQL-kompatiblen Clients— Stellen Sie mithilfe verschiedener Clients und Tools eine Verbindung zu Aurora DSQL her

Generative KI für Aurora DSQL

Dieser Abschnitt enthält detaillierte Anweisungen zur Verwendung von Generative KI-Tools mit Aurora DSQL

AWS Labs Aurora DSQL-MCP-Server

Ein MCP-Server (Model Context Protocol) von AWS Labs für Aurora DSQL

Features

- Konvertierung von menschenlesbaren Fragen und Befehlen in strukturierte Postgres-kompatible SQL-Abfragen und deren Ausführung anhand der konfigurierten Aurora DSQL-Datenbank.
- Standardmäßig schreibgeschützt, Transaktionen aktiviert mit `--allow-writes`
- Wiederverwendung von Verbindungen zwischen Anfragen zur Verbesserung der Leistung
- Integrierter Zugriff auf Aurora DSQL-Dokumentation, Suche und Best-Practice-Empfehlungen

Verfügbare Tools

Datenbankoperationen

- readonly_query — Führt schreibgeschützte SQL-Abfragen für Ihren DSQL-Cluster aus
- transact — Führt Schreiboperationen in einer Transaktion aus (erfordert) `--allow-writes`
- get_schema — Ruft Informationen zum Tabellenschema ab

Dokumentation und Empfehlungen

- dsql_search_documentation — Durchsucht die Aurora DSQL-Dokumentation
  - Parameter: `search_phrase` (erforderlich), (optional) `limit`
- dsql_read_documentation — Liest spezifische DSQL-Dokumentationsseiten
  - Parameter: `url` (erforderlich), (optional), `start_index` (optional) `max_length`
- dsql_recommend — Holen Sie sich Empfehlungen für bewährte DSQL-Praktiken
  - Parameter: (erforderlich) `url`

Voraussetzungen

1. Ein AWS-Konto mit einem Aurora DSQL-Cluster
2. Dieser MCP-Server kann nur lokal auf demselben Host wie Ihr LLM-Client ausgeführt werden.
3. AWS-Anmeldeinformationen mit Zugriff auf AWS-Services einrichten
   - Sie benötigen ein AWS-Konto mit einer Rolle mit diesen Berechtigungen:
     - `dsql:DbConnectAdmin`- Stellen Sie als Admin-Benutzer eine Verbindung zu DSQL-Clustern her

  - `dsql:DbConnect`- Connect zu DSQL-Clustern mit benutzerdefinierten Datenbankrollen her (nur erforderlich, wenn Sie Benutzer ohne Administratorrechte verwenden)
- AWS-Anmeldeinformationen mit `aws configure` oder Umgebungsvariablen konfigurieren

Installation

Für die meisten Tools sollte es ausreichend sein, die Konfiguration anhand der Anweisungen zur Standardinstallation zu aktualisieren.

Für Claude Code und Codex werden separate Anweisungen beschrieben.

Standardinstallation: Aktualisierung der entsprechenden MCP-Konfigurationsdatei

Verwenden von **uv**

1. Installieren Sie es `uv` von Astral oder der GitHub README-Datei
2. Installieren Sie Python mit `uv python install 3.10`

Konfigurieren Sie den MCP-Server in Ihrer MCP-Client-Konfiguration (Suchen der MCP-Konfigurationsdatei)

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "command": "uvx",
      "args": [
        "awslabs.aurora-dsql-mcp-server@latest",
        "--cluster_endpoint",
        "[your dsql cluster endpoint, e.g. abcdefghijklmnopqrst234567.dsql.us-
east-1.on.aws]",
        "--region",
        "[your dsql cluster region, e.g. us-east-1]",
        "--database_user",
        "[your dsql username, e.g. admin]",
        "--profile",
        "[your aws profile, e.g. default]"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR"
      },
```

```
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

Installation von Windows

Für Windows-Benutzer unterscheidet sich das MCP-Serverkonfigurationsformat geringfügig:

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "disabled": false,
      "timeout": 60,
      "type": "stdio",
      "command": "uv",
      "args": [
        "tool",
        "run",
        "--from",
        "awslabs.aurora-dsql-mcp-server@latest",
        "awslabs.aurora-dsql-mcp-server.exe"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR",
        "AWS_PROFILE": "your-aws-profile",
        "AWS_REGION": "us-east-1"
      }
    }
  }
}
```

Die MCP-Client-Konfigurationsdatei finden

Für einige der gängigsten Agentic-Entwicklungstools finden Sie Ihre MCP-Client-Konfigurationen unter den folgenden Dateipfaden:

- Kiro:
  - Benutzerkonfiguration: `~/.kiro/settings/mcp.json`
  - Workspace-Konfiguration: `/path/to/workspace/.kiro/settings/mcp.json`

- Claude Code: Ausführliche Hilfe zur Einrichtung finden Sie unter Claude Code Installation
  - Benutzerkonfiguration: `~/.claude.json` in `"mcpServers"`
  - Projektkonfiguration: `/path/to/project/.mcp.json`
  - Lokale Config: `~/.claude.json` in `"projects" -> "path/to/project" -> "mcpServers"`
- Cursor:
  - Global: `~/.cursor/mcp.json`
  - Projekt: `/path/to/project/.cursor/mcp.json`
- Kodex: `~/.codex/config.toml`
  - Jeder MCP-Server ist mit einer `[mcp_servers.<server-name>]` Tabelle in der Konfigurationsdatei konfiguriert. Weitere Informationen finden Sie in den Installationsanweisungen für Custom Codex
- Warp:
  - Bearbeitung von Dateien: `~/.warp/mcp_settings.json`
  - Anwendungseditor: `Settings > AI > Manage MCP Servers` und fügen Sie JSON ein
- Amazon Q Entwickler-CLI: `~/.aws/amazonq/mcp.json`
- Cline: Normalerweise ein verschachtelter VS-Codepfad - `~/.vscode-server/path/to/cline_mcp_settings.json`

Claude Code

Voraussetzungen

Wichtig: Die MCP-Serververwaltung ist nur über das Claude Code CLI-Terminal-Erlebnis verfügbar, nicht über den systemeigenen Panel-Modus von VS Code.

Installieren Sie zuerst die Claude Code CLI, indem Sie dem von Claude empfohlenen systemeigenen Installationsprozess folgen.

Auswahl des richtigen Bereichs

Claude Code bietet 3 verschiedene Gültigkeitsbereiche: lokal (Standard), Projekt und Benutzer. Außerdem gibt er an, welcher Bereich je nach Vertraulichkeit der Zugangsdaten ausgewählt und geteilt werden muss. Weitere Informationen finden Sie in der Claude Code-Dokumentation zu MCP-Installationsbereichen.

1. Server mit lokalem Geltungsbereich stellen die Standardkonfigurationsebene dar und werden unter dem Pfad Ihres Projekts gespeichert. `~/.claude.json` Sie sind beide für Sie privat und nur

innerhalb des aktuellen Projektverzeichnisses zugänglich. Dies ist die Standardeinstellung `scope` beim Erstellen von MCP-Servern.

2. Projektbezogene Server ermöglichen die Zusammenarbeit im Team und sind dennoch nur in einem Projektverzeichnis zugänglich. Server mit Projektbereich fügen eine `.mcp.json` Datei im Stammverzeichnis Ihres Projekts hinzu. Diese Datei ist so konzipiert, dass sie in die Versionskontrolle eingecheckt werden kann, um sicherzustellen, dass alle Teammitglieder Zugriff auf dieselben MCP-Tools und -Dienste haben. Wenn Sie einen projektbezogenen Server hinzufügen, erstellt oder aktualisiert Claude Code diese Datei automatisch mit der entsprechenden Konfigurationsstruktur.
3. Server mit Benutzerzugriff werden gespeichert **`~/.claude.json`** und bieten projektübergreifenden Zugriff, sodass sie für alle Projekte auf Ihrem Computer verfügbar sind und gleichzeitig für Ihr Benutzerkonto privat bleiben.

Verwenden der Claude-CLI (empfohlen)

Die Verwendung einer interaktiven `claude` CLI-Sitzung ermöglicht eine verbesserte Problembehebung, daher ist dies der empfohlene Weg.

```
claude mcp add amazon-aurora-dsql \
  --scope [one of local, project, or user] \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Fehlerbehebung: Claude Code mit Bedrock auf einem anderen AWS-Konto verwenden

Wenn Sie Claude Code mit einem Bedrock AWS-Konto oder -Profil konfiguriert haben, das sich von dem Profil unterscheidet, das für die Verbindung mit Ihrem DSQL-Cluster benötigt wird, müssen Sie zusätzliche Umgebungsargumente angeben:

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Direkte Änderung in der Konfigurationsdatei

Claude Code Erfordert eine alphanumerische Benennung, daher empfehlen wir, Ihren Server zu benennen:`aurora-dsql-mcp-server`.

Lokaler Geltungsbereich

Update `~/.claude.json` im `mcpServers` projektspezifischen Bereich:

```
{
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Umfang des Projekts

Update `/path/to/project/root/.mcp.json` im Bereich: `mcpServers`

```
{
  "mcpServers": {}
}
```

Nutzer-Geltungsbereich

Aktualisierung `~/.claude.json` innerhalb des projektspezifischen `mcpServers` Feldes:

```
{
  "mcpServers": {}
}
```

Kodex

Option 1: Codex CLI

Wenn Sie die Codex CLI installiert haben, können Sie den Befehl codex mcp verwenden, um Ihre MCP-Server zu konfigurieren.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Option 2: config.toml

Für eine genauere Steuerung der MCP-Serveroptionen können Sie die Konfigurationsdatei manuell bearbeiten. `~/.codex/config.toml` Jeder MCP-Server ist mit einer `[mcp_servers.<server-name>]` Tabelle in der Konfigurationsdatei konfiguriert.

```
[mcp_servers.amazon-aurora-dsql]
command = "uvx"
args = [
  "awslabs.aurora-dsql-mcp-server@latest",
  "--cluster_endpoint", "<DSQL_CLUSTER_ID>.dsql.<AWS_REGION>.on.aws",
  "--region", "<AWS_REGION>",
  "--database_user", "<DATABASE_USERNAME>"
]

[mcp_servers.amazon-aurora-dsql.env]
FASTMCP_LOG_LEVEL = "ERROR"
```

Installation wird überprüft

Für Amazon Q Developer CLI, Kiro CLI, ClaudeCLI/TUI, or Codex CLI/TUI, ausführen, `/mcp` um den Status des MCP-Servers zu sehen.

Für die Kiro IDE können Sie auch zur `MCP SERVERS` Registerkarte des Kiro Panels navigieren, auf der alle konfigurierten MCP-Server und ihre Verbindungsstatusanzeigen angezeigt werden.

Optionen für die Serverkonfiguration

`--allow-writes`

Standardmäßig erlaubt der DSQL-MCP-Server keine Schreiboperationen („Nur-Lese-Modus“). Alle Aufrufe des Transact-Tools schlagen in diesem Modus fehl. Um das Transact-Tool zu verwenden, lassen Sie Schreibvorgänge zu, indem Sie den Parameter übergeben. `--allow-writes`

Wir empfehlen, bei der Verbindung zu DSQL den Zugriff mit den geringsten Rechten zu verwenden. Beispielsweise sollten Benutzer nach Möglichkeit eine Rolle verwenden, die schreibgeschützt ist. Im Modus „Nur lesen“ wird nach bestem Wissen und Gewissen auf der Clientseite versucht, Mutationen zurückzuweisen.

`--cluster_endpoint`

Dies ist ein obligatorischer Parameter, um den Cluster anzugeben, zu dem eine Verbindung hergestellt werden soll. Dies sollte der vollständige Endpunkt Ihres Clusters sein, z. B. `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`

`--database_user`

Dies ist ein obligatorischer Parameter, um den Benutzer anzugeben, als der eine Verbindung hergestellt werden soll. Zum Beispiel `admin` oder `my_user`. Beachten Sie, dass die AWS-Anmeldeinformationen, die Sie verwenden, über die Berechtigung verfügen müssen, sich als dieser Benutzer anzumelden. Weitere Informationen zum Einrichten und Verwenden von Datenbankrollen in DSQL finden Sie unter Datenbankrollen mit IAM-Rollen verwenden.

`--profile`

Sie können das AWS-Profil angeben, das für Ihre Anmeldeinformationen verwendet werden soll. Beachten Sie, dass dies für die Docker-Installation nicht unterstützt wird.

Die Verwendung der `AWS_PROFILE` Umgebungsvariablen in Ihrer MCP-Konfiguration wird ebenfalls unterstützt:

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Wenn keines angegeben ist, verwendet der MCP-Server standardmäßig das „Standard“ -Profil in Ihrer AWS-Konfigurationsdatei.

`--region`

Dies ist ein obligatorischer Parameter, um die Region Ihrer DSQL-Datenbank anzugeben.

`--knowledge-server`

Optionaler Parameter zur Angabe des Remote-MCP-Serverendpunkts für DSQL-Wissenstools (Suchen, Lesen und Empfehlungen in der Dokumentation). Standardmäßig ist er vorkonfiguriert.

Beispiel:

```
--knowledge-server https://custom-knowledge-server.example.com
```

Hinweis: Verwenden Sie aus Sicherheitsgründen nur vertrauenswürdige Knowledge Server-Endpunkte. Der Server sollte ein HTTPS-Endpunkt sein.

`--knowledge-timeout`

Optionaler Parameter zur Angabe des Timeouts in Sekunden für Anfragen an den Knowledge Server.

Standard: `30.0`

Beispiel:

```
--knowledge-timeout 60.0
```

Erhöhen Sie diesen Wert, wenn beim Zugriff auf Dokumentation in langsamen Netzwerken Timeouts auftreten.

Aurora DSQL Steering: Fähigkeiten und Fähigkeiten

In diesem Abschnitt wird beschrieben, wie die KI-Steuerung für Aurora DSQL mithilfe von Fähigkeiten und Befugnissen konfiguriert wird. Diese auf Markdown basierenden Konfigurationsdateien bieten Kontext und Anleitungen, die KI-Assistenten bei der Codegenerierung automatisch anwenden, um die Qualität der Agentenentwicklung zu verbessern.

-Übersicht

Fähigkeiten und Fähigkeiten sind modulare Funktionen, die die Funktionen des KI-Assistenten für Aurora DSQL erweitern. Sie verpacken Anweisungen, Metadaten und Ressourcen, die KI-Assistenten bei der Arbeit mit Aurora DSQL-Datenbanken automatisch verwenden.

Warum Fähigkeiten und Fähigkeiten einsetzen

Fähigkeiten und Fähigkeiten bieten mehrere wichtige Vorteile für die Entwicklung von Aurora DSQL:

- Spezialisierte KI-Assistenten — Stellen domänenspezifisches Fachwissen für Aurora DSQL bereit, einschließlich Best Practices, Postgres-kompatibler SQL-Muster und verteilter Datenbankoptimierungen.
- Weniger Wiederholungen — Einmal erstellen, automatisch verwenden. Macht es überflüssig, in mehreren Konversationen immer wieder dieselbe Anleitung zu geben.
- Kontexteffizienz — Fähigkeiten werden auf Abruf geladen, anstatt den Kontext im Voraus zu verbrauchen. Die KI lädt Informationen nach Bedarf stufenweise.
- Kontinuierliches Lernen — Im Zuge der Weiterentwicklung der DSQL-Funktionen von Aurora greifen KI-Assistenten automatisch auf aktualisierte Muster zu, wenn die Fähigkeiten aktualisiert werden.

Empfohlene Setup-Pfade

Wählen Sie den Setup-Pfad, der zu Ihrer Entwicklungsumgebung passt:

- the section called “Fähigkeiten CLI”(Agentenunabhängig)
- the section called “Kiro Power”
- the section called “Claude Skill”
- the section called “Zwillingsfähigkeit”
- the section called “Codex-Fähigkeit”

Der DSQL-Skill kann auch mit anderen KI-Codierungsagenten verwendet werden, indem der Skill-Ordner in das Verzeichnis oder des Tools kopiert wird. `rules skills`

Fähigkeiten CLI

Der DSQL-Skill kann mit der Skills CLI installiert werden. Diese agentenunabhängige Einrichtungsmethode funktioniert mit den meisten KI-Codierungsassistenten und ermöglicht es Ihnen, den Skill für mehrere Agenten gleichzeitig zu installieren.

Einrichtung

Führen Sie den folgenden Befehl aus, um den Aurora DSQL-Skill zu installieren:

```
npx skills add awslabs/mcp --skill dsql
```

Die CLI führt Sie durch:

- Agenten auswählen — Wählen Sie aus, auf welchen Agenten die Installation erfolgen soll (Kiro, Claude Code, Cursor, Copilot, Gemini, Codex, Roo, Cline,, Windsurf usw.) OpenCode
- Umfang der Installation — Wählen Sie zwischen:
  - Projekt: Im aktuellen Verzeichnis installieren (mit Ihrem Projekt übergeben)
  - Global: Im Home-Verzeichnis installieren (in allen Projekten verfügbar)
- Installationsmethode — Wählen Sie zwischen:
  - Symlink (empfohlen): Zentrale Informationsquelle, einfache Updates
  - An alle Agenten kopieren: Unabhängige Kopien für jeden Agenten

Verwalten von Qualifikationen

Sie können Ihre Fähigkeiten jederzeit überprüfen und aktualisieren, indem Sie:

```
npx skills check
npx skills update
```

Kiro Power

Bei Kiro Powers handelt es sich um vereinheitlichte Pakete, die MCP-Tools mit Framework-Fachwissen und Steuerungsanweisungen kombinieren. Jede Leistung umfasst ein Einstiegsdokument, in dem die verfügbaren MCP-Tools und Aktivierungsauslöser, die MCP-Serverkonfiguration sowie zusätzliche, auf Anfrage bereitgestellte Workflow-spezifische Anleitungen erläutert werden.

Die Befugnisse werden dynamisch auf der Grundlage des Benutzerkontextes aktiviert. Anstatt alle Tools im Voraus zu laden, halten Powers die Basisnutzung bei nahezu Null, bis die Aktivierung durch relevante Schlüsselwörter ausgelöst wird.

Einrichtung

So richten Sie die Kiro-Stromversorgung für Aurora DSQL ein:

1. Direkt aus der Kiro Powers Registry installieren
2. Sobald Sie zu Power in der IDE umgeleitet wurden, gehen Sie entweder wie folgt vor:
   - Wählen Sie die Taste Try Power aus. Empfohlen für Benutzer, die möchten, dass die KI die Einrichtung des MCP-Servers steuert oder ein interaktives Onboarding-Erlebnis mit Aurora DSQL ermöglicht, um einen neuen Cluster zu erstellen.

- Öffne einen neuen Kiro-Chat und frage nach allem, was mit Aurora DSQL zu tun hat. Aktualisieren Sie optional die MCP-Konfiguration mit Ihren vorhandenen Clusterdetails, um die MCP-Serververbindung zu testen, sodass sie sofort mit Strom verwendet werden kann. Der Kiro-Agent aktiviert die Stromversorgung automatisch, wenn er feststellt, dass sie für die Ausführung der Aufgabe des Benutzers wertvoll ist.

Claude Skill

Claude-Skills sind modulare Fähigkeiten, die die Funktionalität von Claude erweitern. Jeder Skill enthält Anweisungen, Metadaten und optionale Ressourcen, die Claude bei Bedarf automatisch verwendet. Skills sind dateisystembasiert und werden bei Bedarf geladen, um den Kontextverbrauch zu minimieren.

Einfache Einrichtung mit der Skills CLI

Der Skill kann mit dem in Claude Code installiert werdenthe section called “Fähigkeiten CLI”. Um nur Claude Code als Agenten für die Installation anzugeben, verwenden Sie:

```
npx skills add awslabs/mcp --skill dsql --agent claude-code
```

Alternative: Direktes Setup mit einem Git-Clone

Das alternative Setup verwendet einen spärlichen Klon des dsql-Skill-Verzeichnisses und verlinkt diesen Klon mit dem Ordner. `~/.claude/skills/` Dadurch können Änderungen am Skill immer dann abgerufen werden, wenn der Skill aktualisiert werden muss.

Voraussetzungen

- Git installiert

Schritte zur Einrichtung

1. Erstellen Sie ein Basis-Repos-Verzeichnis

```
mkdir -p .dsql_skill_repos
```

2. Sparse klone den Skill aus dem MCP-Repository

Klonen Sie nur den `dsql-skill` Ordner (keine anderen Dateien):

```
cd .dsql_skill_repos
git clone --filter=blob:none --no-checkout https://github.com/awslabs/mcp.git
cd mcp
git sparse-checkout init --cone
git sparse-checkout set src/aurora-dsql-mcp-server/skills/dsql-skill
git checkout
cd ../..
```

3. Verknüpfen Sie den Skill mit dem Skills-Verzeichnis

Fügen Sie das Skills-Verzeichnis hinzu (Standard: global/user-scoped):

```
mkdir -p ~/.claude/skills
```

> **Note**
>
> Wenn Sie diesen Skill zu einem projektbezogenen Skill machen möchten, verwenden Sie stattdessen das Stammverzeichnis Ihres Projekts. `.claude/skills/`

Fügen Sie den Symlink hinzu:

```
ln -s "$(pwd)/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/dsql-skill"
 ~/.claude/skills/dsql-skill
```

4. Überprüfen Sie das Setup

```
# Should show SKILL.md and other skill files
ls -la ~/.claude/skills/dsql-skill/
```

5. Überprüfen Sie den Einsatz von Fähigkeiten

Sobald der Skill konfiguriert ist, sollten Sie einen neuen Skill-Befehl haben:`/dsql`. Möglicherweise müssen Sie Claude Code nach dem Hinzufügen des Skills neu starten, damit er erkannt wird. Sie können diesen Befehl nach Bedarf über die CLI oder das Panel von Claude Code verwenden.

Den Skill aktualisieren

Um die neuesten Änderungen aus dem Repository abzurufen:

```
cd .dsql_skill_repos/mcp
git pull
```

Verzeichnisstruktur

Nachdem Sie einen globalen Skill eingerichtet haben, sollten Sie die folgenden Verzeichnisse sehen:

```
.dsql_skill_repos/
### mcp/                              # Sparse git checkout
    ### src/
        ### aurora-dsql-mcp-server/
            ### skills/
                ### dsql-skill/
                    ### SKILL.md
                    ### ...

~/.claude/
### skills/
    ### dsql-skill -> /path/to/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/
dsql-skill
```

> **Note**
>
> Fügen Sie `.dsql_skill_repos/` es zu Ihrem hinzu, `.gitignore` wenn Sie es nicht verfolgen möchten. Beim Sparse-Checkout wird nur der Skill-Ordner beibehalten, wodurch der Festplattenverbrauch minimiert wird.

Gemini-Fähigkeit

Um den Aurora DSQL-Skill direkt in Gemini hinzuzufügen, entscheiden Sie sich für einen Bereich: `workspace` (im Projekt enthalten) oder `user` (Standard, global) und verwenden Sie das Skills-Installationsprogramm.

Einrichtung

```
gemini skills install https://github.com/awslabs/mcp.git --path src/aurora-dsql-mcp-server/skills/dsql-skill --scope $SCOPE
```

Ersetze es entweder `$SCOPE` durch oder`workspace`. `user`

Sie können dann den `/dsql` Skill-Befehl mit Gemini verwenden, und Gemini erkennt automatisch, wann der Skill verwendet werden sollte.

Codex-Fähigkeit

Verwenden Sie den Skill-Installer von der Codex-CLI oder der TUI aus, indem Sie den `$skill-installer` Skill verwenden.

Einrichtung

```
$skill-installer install dsql skill: https://github.com/awslabs/mcp/tree/main/src/
aurora-dsql-mcp-server/skills/dsql-skill
```

Starte Codex neu, um den Skill zu erwerben. Die Fähigkeit kann dann aktiviert werden mit`$dsql`.

Erste Schritte mit dem Aurora DSQL Query Editor

Mit dem Aurora DSQL Query Editor können Sie eine sichere Verbindung zu Ihren Aurora DSQL-Clustern herstellen und SQL-Abfragen direkt von der AWS Management Console aus ausführen, ohne externe Clients installieren oder konfigurieren zu müssen. Es bietet einen intuitiven Arbeitsbereich mit integrierter Syntaxhervorhebung, automatischer Vervollständigung und intelligenter Codeunterstützung. Sie können Schemaobjekte schnell untersuchen, SQL-Abfragen entwickeln und ausführen und Ergebnisse anzeigen — alles über eine einzige Oberfläche.

In diesem Thema erfahren Sie, wie Sie eine Verbindung zu einem Cluster herstellen, Abfragen ausführen, Ergebnisse anzeigen und erweiterte Funktionen wie Ausführungspläne kennenlernen.

> **Note**
>
> Der Abfrage-Editor ist in allen Regionen verfügbar, in denen Aurora DSQL unterstützt wird. Weitere Informationen zur regionalen Verfügbarkeit finden Sie unter AWS Regionale Dienste.

Voraussetzungen

Überprüfen Sie zu Beginn, ob die folgenden Anforderungen erfüllt sind:

- Sie haben mindestens einen Aurora DSQL-Cluster verfügbar. Weitere Informationen zum Erstellen von Clustern finden Sie unterSchritt 1: Erstellen eines Einzelregions-Clusters mit Aurora DSQL.

- Ihr Cluster-Endpunkt ist öffentlich zugänglich. Der Abfrage-Editor unterstützt keine Cluster, deren öffentlicher Zugriff durch ressourcenbasierte Richtlinien blockiert wird, oder Cluster, die über VPC-Endpunkte verwaltet werden. Weitere Informationen zu Zugriffsbeschränkungen finden Sie unter und. Blockieren des öffentlichen Zugriffs mit ressourcenbasierten Richtlinien in Aurora DSQL Verwaltung und Verbindung zu Amazon Aurora DSQL-Clustern mithilfe AWS PrivateLink
- Ihr IAM-Benutzer oder Ihre IAM-Rolle verfügt über die erforderlichen Berechtigungen, um auf den Cluster zuzugreifen und eine Verbindung zu diesem herzustellen. Weitere Informationen zu Berechtigungen finden Sie unterVerwenden von Datenbankrollen und IAM-Authentifizierung.

Arbeiten mit dem Abfrage-Editor

Öffnen Sie den Abfrage-Editor

Um den Abfrage-Editor zu öffnen

1. Öffnen Sie die Aurora DSQL-Konsole.
2. Wählen Sie im Navigationsbereich die Option Abfrage-Editor aus.

Wählen Sie alternativ auf der Seite Cluster den Cluster aus, den Sie abfragen möchten, und wählen Sie Mit dem Abfrage-Editor Connect, um den Editor direkt zu starten.

> **Note**
>
> Arbeit und Verbindungsstatus werden nicht gespeichert. Wenn Sie die Aurora DSQL-Konsole verlassen, den Browser-Tab schließen oder sich abmelden, gehen Ihre Verbindungen, der Abfragetext und die Ergebnisse verloren.

Verbinden mit einem Cluster

Um eine Verbindung zu einem Cluster herzustellen

1. Wenn keine Clusterverbindung besteht, zeigt der Editor die Meldung Kein Cluster wurde verbunden an. Wählen Sie Connect oder wählen Sie + (Hinzufügen) im Cluster-Explorer-Bereich, um eine Verbindung zu einem vorhandenen Cluster herzustellen.
2. (Optional) Stellen Sie mithilfe verschiedener Rollen eine Verbindung zu mehreren Clustern oder zu demselben Cluster her.

Erkunden Sie Cluster-Objekte

Der Cluster-Explorer zeigt alle verfügbaren Cluster-Verbindungen an und ermöglicht das Durchsuchen von Objekten wie Datenbanken, Schemas, Tabellen und Ansichten. Er bietet auch allgemeine Aktionen wie Aktualisieren, Tabelle erstellen und andere kontextspezifische Optionen.

Abfragen ausführen

Eine Abfrage ausführen

1. Geben Sie im Tabbereich des Abfrage-Editors Ihre SQL-Anweisung ein. Beispiel:

   ```
   SELECT * FROM public.orders LIMIT 10;
   ```

2. Überprüfen Sie den Active Cluster-Kontext, der oben rechts auf der Registerkarte „Abfrage“ angezeigt wird. Dies gibt die Clusterverbindung an, die der aktuellen Abfrageregisterkarte zugeordnet ist.
3. (Optional) Verwenden Sie das Verbindungs-Dropdown-Menü, um alle verfügbaren Verbindungen zu überprüfen oder zu einem anderen Cluster zu wechseln. Wenn Sie die Verbindung ändern, wird dort aktualisiert, wo Ihre Abfragen auf dieser Registerkarte ausgeführt werden.
4. Wählen Sie dann Ausführen aus, um die Abfrage auszuführen.

> **Note**
>
> Jede Abfrage kann bis zu 10.000 Zeilen im Ergebnisbereich zurückgeben. Bei größeren Datensätzen verfeinern Sie Ihre Abfrage mit Filtern oder Grenzwerten.

Überprüfen Sie die Ergebnisse und Ausführungspläne

Nachdem die Abfrage ausgeführt wurde, überprüfen Sie die Ausgabe im Ergebnisbereich unten im Editor. Standardmäßig wird bei jeder Abfrageausführung die Registerkarte Ergebnisse (Tabelle) angezeigt, auf der die tabellarische Abfrageausgabe angezeigt wird.

Um den Ausführungsplan für die Abfrage abzurufen, führen Sie den Befehl `EXPLAIN ANALYZE` oder aus, `EXPLAIN ANALYZE VERBOSE` um zusätzliche Einblicke in die Abfrageleistung zu erhalten. Weitere Informationen zu Ausführungsplänen finden Sie unterAurora DSQL EXPLAIN-Pläne lesen.

> **Tip**
>
> Der `EXPLAIN ANALYZE VERBOSE` Befehl zeigt Schätzungen zur DPU-Nutzung an, einschließlich der Werte für Compute, Read, Write und Total DPU, sodass sofort ersichtlich ist, welche Ressourcen von einzelnen SQL-Anweisungen verbraucht werden.

Abfrage-Editoren: Verwendung JupyterLab mit Aurora DSQL

Dieses Handbuch enthält step-by-step Anweisungen zur Verbindung und Abfrage von Amazon Aurora DSQL JupyterLab mithilfe von Python. JupyterLab ist eine beliebte interaktive Computerumgebung, die Code, Text und Visualisierungen in einem einzigen Dokument kombiniert. Sie wird häufig für datenwissenschaftliche und Forschungsanwendungen verwendet.

Die folgenden Anweisungen behandeln die Grundlagen der Verwendung von Aurora DSQL sowohl bei einer lokalen Installation JupyterLab als auch bei der Verwendung von Amazon SageMaker AI, einem vollständig verwalteten Service für maschinelles Lernen, der eine gehostete Umgebung mit einer Benutzeroberfläche für Datenworkflows bietet.

Erste Schritte

Voraussetzungen

- Ein Aurora DSQL-Cluster
- AWS-Anmeldeinformationen konfiguriert (nur lokale Installation)
- Python-Version 3.9 oder höher (nur lokale Installation)

Lokal verwenden JupyterLab

Um damit zu beginnen JupyterLab, müssen Benutzer zuerst die Anwendung mit Pythons Pip installieren:

```
pip install jupyterlab
```

JupyterLab kann dann durch Ausführen **`jupyter lab`** geöffnet werden. Dadurch wird die JupyterLab Anwendung unter localhost:8888 geöffnet, auf die in einem Browser zugegriffen werden kann. Stellen Sie sicher, dass Sie AWS-Anmeldeinformationen in Ihrer lokalen Umgebung konfiguriert haben, bevor Sie fortfahren.

Amazon SageMaker AI verwenden

Gehen Sie in der AWS-Konsole zur Amazon SageMaker AI-Konsolenseite und dann zum Abschnitt Notebooks unter Anwendungen und IDEs. Von dort aus können Sie Notebook-Instance erstellen auswählen, um mit der Erstellung einer SageMaker Umgebung zu beginnen. Wählen Sie einen Instanztyp und eine Plattform aus, bevor Sie auf Notebook-Instanz erstellen klicken.

Weitere Informationen zur Einrichtung und zu den Instance-Optionen finden Sie in der Amazon SageMaker AI-Setup-Dokumentation.

> **Note**
>
> Warnung: Die Verwendung von Amazon SageMaker AI kann zu Gebühren für Ihr AWS-Konto führen.

Sobald die SageMaker Instance aktiv ist, können Sie sie im Bereich Notebook-Instances mit Öffnen öffnen JupyterLab. Bevor Sie mit Aurora DSQL in Ihrem Notebook beginnen, müssen Sie in der IAM-Rolle der SageMaker Instanz Zugriff auf Ihren DSQL-Cluster gewähren. Der einfachste Weg, dies zu tun, besteht darin, dem Link zur IAM-Rolle auf der Notebook-Instanzseite zu folgen. Von dort aus können Sie die mit Ihrer SageMaker IAM-Rolle verknüpften Richtlinien bearbeiten. Weitere Informationen zur Konfiguration einer IAM-Richtlinie für den Zugriff auf Aurora DSQL finden Sie unter Authentifizierung und Autorisierung.

Verbindung zu Aurora DSQL herstellen mit JupyterLab

Nachdem Sie eine JupyterLab Instanz eingerichtet haben, sind die Schritte zum Herstellen einer Verbindung mit Aurora DSQL lokal und in SageMaker AI identisch. Erstellen Sie ein leeres Python 3-Notizbuch, in dem Sie Zellen mit Python-Code hinzufügen können.

Laden Sie in einer Python-Zelle das Amazon-Stammzertifikat aus dem offiziellen Trust Store herunter:

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Um eine Verbindung zu Aurora DSQL herzustellen, installieren Sie zuerst den Aurora DSQL Connector für Python und den Psycopg-Treiber in einer Python-Zelle und importieren Sie ihn dann:

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Nachdem der Konnektor importiert wurde, können Sie dann eine DSQL-Konfiguration erstellen und eine Verbindung herstellen. Der Aurora DSQL Python Connector übernimmt automatisch die Erstellung eines Authentifizierungstokens für jede Verbindung.

```
config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin"
    }

conn = dsql.connect(**config)
```

Nachdem Sie Ihren Code ausgeführt haben, sollten Sie jetzt eine Psycopg-Verbindung zu Aurora DSQL haben. Sie können dann Abfragen mit dem Psycopg-Cursor ausführen und Ihre SQL-Abfrage angeben. Weitere Informationen zur Verwendung von Psycopg mit einer Postgres-kompatiblen Datenbank finden Sie in der Psycopg-Dokumentation. Diese Abfrage führt zu einer Liste von Tupeln in. `results_list`

```
with conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

Sie können dann Python-Frameworks wie Pandas verwenden, um Ihre Abfrageergebnisse zu analysieren oder zu visualisieren, zum Beispiel:

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

Beispiel für ein Notizbuch

Ein Beispielnotizbuch mit Aurora DSQL ist im Aurora DSQL Samples Repository verfügbar.

Weitere Informationen

Dokumentation zur Einrichtung von Amazon SageMaker AI

Aurora DSQL-Konnektor für Python

Pandas-Dokumentation

Backup und Wiederherstellung für Amazon Aurora DSQL

Amazon Aurora DSQL unterstützt Sie dabei, Ihre Anforderungen an regulatorische Compliance und Geschäftskontinuität zu erfüllen – durch die Integration mit AWS Backup, einem vollständig verwalteten Datenschutzdienst, mit dem Backups über AWS hinweg, in der Cloud und On-Premises zentralisiert und automatisiert. Der Service optimiert die Erstellung, Verwaltung und Wiederherstellung von Backups für Aurora DSQL-Cluster in einer oder mehreren Regionen.

Nachstehend sind einige der wichtigsten Funktionen aufgelistet:

- Zentralisiertes Backup-Management über die AWS-Managementkonsole, SDK oder über AWS CLI
- Vollständige Cluster-Backups
- Automatisierte Sicherungszeitpläne und Aufbewahrungsrichtlinien
- Regions- und kontoübergreifende Funktionen
- WORM-Konfiguration (Write Once, Read Many) für alle gespeicherten Backups

Weitere Informationen zu den Features von AWS Backup-Vault Lock sowie eine umfangreiche Liste der verfügbaren AWS Backup-Features für Aurora DSQL finden Sie unter Vorteile der Tresorsperre und Verfügbarkeit von AWS Backup-Features im AWS BackupEntwicklerhandbuch.

Erste Schritte mit AWS Backup

AWS Backup erstellt vollständige Kopien Ihrer Aurora DSQL-Cluster. Beginnen Sie Ihre Verwendung von AWS Backup für Aurora DSQL, indem Sie den Anleitungen unter Erste Schritte mit AWS Backup folgen:

1. Erstellen von On-Demand-Backups für sofortigen Schutz
2. Erstellen von Backup-Plänen für automatisierte, geplante Backups
3. Konfigurieren von Aufbewahrungszeiträumen und regionsübergreifendem Kopieren
4. Einrichten der Überwachung und Benachrichtigungen für Backup-Aktivitäten

Wiederherstellen Ihrer Backups

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

Wiederherstellen von Clustern für eine einzelne Region

Um einen Aurora DSQL-Cluster für eine einzelne Region 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 Anleitungen finden Sie unter Wiederherstellen eines Aurora DSQL-Clusters einer einzelnen Region.

Wiederherstellen von Clustern mit mehreren Regionen

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

Für die Wiederherstellung eines Aurora DSQL-Clusters mit mehreren Regionen können Sie ein Backup verwenden, das in einer einzigen AWS-Region erstellt wurde. Bevor Sie den Wiederherstellungsvorgang einleiten, müssen Sie zunächst sicherstellen, dass eine identische Kopie Ihres Backups in allen AWS-Regionen für Ihre Cluster mit mehreren Regionen vorhanden ist. Wenn Sie diese Kopien noch nicht haben, müssen Sie das Backup zuerst auf eine andere AWS-Region kopieren, die Cluster für mehrere Regionen unterstützt.

Wir empfehlen, Sicherungskopien in wichtigen AWS-Regionen zu erstellen, um robuste Notfallwiederherstellungsoptionen zu ermöglichen und Compliance-Anforderungen zu erfüllen. Verfügbare AWS-Regionen für Aurora DSQL können Sie unter the section called “AWS-Region Verfügbarkeit” einsehen.

Detaillierte Anleitungen zu diesen Schritten finden Sie in der Dokumentation zu Amazon Aurora DSQL-Wiederherstellung.

Überwachung und Compliance

AWS Backup bietet umfassende Einblicke in Sicherungs- und Wiederherstellungsvorgänge unter Zuhilfenahme der folgenden Ressourcen.

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

Weitere Informationen zur Protokollierung von Aktionen, die von einem Benutzer, einer Rolle oder einem AWS-Service bei der Verwendung von Aurora DSQL durchgeführt werden, finden Sie unter Protokollierung von Aurora DSQL-Vorgängen mit AWS CloudTrail.

Weitere Ressourcen

Weitere Informationen über AWS Backup-Features und deren Verwendung in Kombination mit Aurora DSQL finden Sie in den folgenden Ressourcen:

- Verwaltete Richtlinien für AWS Backup
- Amazon Aurora DSQL Wiederherstellung
- Unterstützte Services nach AWS-Region
- Backup-Verschlüsselung in AWS Backup

Der Einsatz von AWS Backup für Aurora DSQL ermöglicht Ihnen die Implementierung einer robusten, konformen und automatisierten Backup-Strategie, die Ihre kritischen Datenbankressourcen schützt und gleichzeitig den Verwaltungsaufwand minimiert. Ganz gleich, 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

Die Überwachung und Protokollierung ist ein wichtiger Teil zur Aufrechterhaltung von Zuverlässigkeit, Verfügbarkeit und Leistung Ihrer Amazon Aurora DSQL-Ressourcen. Sie sollten Überwachungs- und Protokolldaten aus allen Bereichen Ihrer Aurora DSQL-Ressourcen erfassen, damit Sie einen Fehler mit mehreren Ursachen problemlos analysieren 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.

Überwachen von Aurora DSQL-Metriken mit Amazon CloudWatch

Sie können Aurora DSQL mit CloudWatch überwachen, wobei Rohdaten erfasst und in lesbare, nahezu Echtzeit-Metriken verarbeitet werden. CloudWatch speichert diese Statistiken für 15 Monate, wodurch Sie ein besseres Verständnis für die Leistung Ihrer Webanwendung oder Ihres Services gewinnen können. Richten Sie Alarme ein, um bestimmte Schwellenwerte zu überwachen und Benachrichtigungen zu senden oder Maßnahmen zu ergreifen, sobald diese erreicht werden. Sehen Sie sich die folgenden Nutzungs- und Beobachtbarkeitsmetriken an, die für Aurora DSQL verfügbar sind.

Weitere Informationen finden Sie im Amazon CloudWatch-Benutzerhandbuch.

Beobachtbarkeit und Leistung

In dieser Tabelle werden Beobachtbarkeitsmetriken für Aurora DSQL beschrieben. Sie enthält Metriken für die Nachverfolgung schreibgeschützter Transaktionen und der Gesamtzahl aller Transaktionen, um eine allgemeine Beschreibung des Workloads 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	Read-only transactions	none	The number of read-only transactions
TotalTransactions	Total transactions	none	The total number of transactions executed on the system, including read-only transactions.
QueryTimeouts	Query timeouts	none	The number of queries which have timed out due to hitting the maximum transaction time
OccConflicts	OCC conflicts	none	The number of transactions aborted due to key level OCC
CommitLatency	Commit Latency	milliseconds	Time spent by commit phase of query execution (P50)
BytesWritten	Bytes Written	bytes	Bytes written to storage
BytesRead	Bytes Read	bytes	Bytes read from storage
ComputeTime	QP compute time	milliseconds	QP wall clock time
ClusterStorageSize	Cluster Storage Size	bytes	Cluster size

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
WriteDPU	Write Units	<cluster-id>	DPU	Approximates the write active-use component of your Aurora DSQL cluster DPU usage.
MultiRegionWriteDPU	Multi-Region Write Units	<cluster-id>	DPU	Applicable for Multi-Region clusters: Approximates the multi-Region write active-use component of your Aurora DSQL cluster DPU usage.
ReadDPU	Read Units	<cluster-id>	DPU	Approximates the read active-use component of your Aurora DSQL cluster DPU usage.
ComputeDPU	Compute Units	<cluster-id>	DPU	Approximates the compute active-use component of

CloudWatch-Metrikname	Metrik	Dimension: ResourceId	Einheit	Beschreibung
				your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

Protokollierung von Aurora DSQL-Vorgängen mit AWS CloudTrail

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

CloudTrail erfasst alle API-Aufrufe für Aurora DSQL als Ereignisse. Aurora DSQL zeichnet Konsolenaktivitäten als Verwaltungsereignisse auf. Auch authentifizierte Verbindungsversuche zu Clustern werden als Datenereignisse erfasst.

Mit den von CloudTrail erfassten Informationen können Sie die an Aurora DSQL gestellte Anfrage, die IP-Adresse, von der die Anfrage ausging, wann die Anfrage gesendet wurde, die Benutzeridentität des Senders und weitere Details bestimmen.

Bei Kontoerstellung ist CloudTrail standardmäßig in Ihrem AWS-Konto aktiviert und Sie haben Zugriff auf den CloudTrail-Ereignisverlauf. Der CloudTrail-Ereignisverlauf stellt eine anzeigbare, durchsuchbare, herunterladbare und unveränderliche Aufzeichnung der Verwaltungsereignisse der letzten 90 Tage in einer AWS-Region bereit. Weitere Informationen finden Sie unter Arbeiten mit dem CloudTrail-Ereignisverlauf im AWS CloudTrail-Benutzerhandbuch. Für die Aufzeichnung des Ereignisverlaufs fallen keine CloudTrail-Gebühren an.

Um eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS-Konto zu erstellen – einschließlich Ereignissen für Aurora DSQL – können Sie entweder einen Trail oder einen Lake-Ereignisdatenspeicher für AWS einrichten (eine zentrale Lösung zur Speicherung und Analyse von AWS CloudTrail-Ereignissen). Weitere Informationen zum Erstellen von Trails finden Sie unter Arbeiten mit CloudTrail-Trails. Weitere Informationen zum Einrichten und Verwalten von Ereignisdatenspeichern finden Sie unter CloudTrail Lake-Ereignisdatenspeicher.

Aurora DSQL-Verwaltungsereignisse in CloudTrail

CloudTrail-Verwaltungsereignisse liefern Informationen zu Verwaltungsvorgängen, die auf Ressourcen in Ihrem AWS-Konto ausgeführt werden. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. Standardmäßig erfasst CloudTrail Verwaltungsereignisse im Ereignisverlauf.

Amazon Aurora DSQL protokolliert alle Aurora DSQL-Operationen auf der Steuerebene als Verwaltungsereignisse. Eine Liste der Amazon Aurora DSQL-Operationen auf der Steuerebene, die Aurora DSQL in CloudTrail protokolliert, finden Sie in der Aurora DSQL API-Referenz.

Protokolle der Steuerebene

Amazon Aurora DSQL protokolliert die folgenden Operationen auf der Steuerebene von Aurora DSQL als Verwaltungsereignisse in CloudTrail.

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

Protokolle zu Backups und Wiederherstellungen

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

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

Weitere Informationen dazu, wie Sie Ihre Aurora DSQL-Cluster mithilfe von AWS Backup schützen finden Sie unter Backup und Wiederherstellung für Amazon Aurora DSQL.

AWS KMS-Protokolle

Amazon Aurora DSQL protokolliert die folgenden AWS KMS-Operationen als Verwaltungsereignisse in CloudTrail.

- `GenerateDataKey`
- `Decrypt`

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

Aurora DSQL-Datenereignisse in CloudTrail

CloudTrail-Datenereignisse liefern typischerweise Informationen zu den Ressourcenoperationen, die für oder auf einer Ressource ausgeführt wurden. Des Weiteren erfassen sie die Operationen des Services auf Datenebene. Datenereignisse sind oft Aktivitäten mit hohem Volume. Standardmäßig protokolliert 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-Managementkonsole und Protokollieren von Datenereignissen mit dem AWS Command Line Interface im AWS CloudTrail-Benutzerhandbuch.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zu CloudTrail-Preisen finden Sie unter AWS CloudTrail – Preise.

Für Aurora DSQL erfasst CloudTrail 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 in der CloudTrail-Konsole aus der Liste Ressourcentyp ausgewählt werden kann. In der Wertspalte resources.type wird der `resources.type` Wert angezeigt, den Sie angeben würden, wenn Sie erweiterte Event-Selektoren mithilfe der APIs AWS CLI oder CloudTrail konfigurieren würden. In der Spalte Daten-APIs, die in CloudTrail protokolliert wurden, werden die API-Aufrufe angezeigt, die für den Ressourcentyp in CloudTrail protokolliert wurden.

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

Sie können erweiterte Ereignis-Selektoren konfigurieren, um anhand der Felder `eventName` und `resources.`ARN gezielt nur gefilterte Ereignisse zu protokollieren. Weitere Informationen zu diesen Kontingenten finden Sie unter AdvancedFieldSelector in derAWS CloudTrail-API-Referenz.

Im folgenden Beispiel sehen Sie, wie Sie AWS CLI zum Konfigurieren von `dsql-data-events-trail` verwenden können, um Datenereignisse für Aurora DSQL zu erhalten.

```
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 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 im Umfang nach Compliance-Programm 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 geteilten Verantwortung bei der Verwendung von Aurora DSQL einsetzen können. In den folgenden Themen wird veranschaulicht, wie Sie Aurora DSQL zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie lernen auch, wie Sie andere AWS Dienste nutzen können, die Ihnen helfen, Ihre Aurora DSQL-Ressourcen zu überwachen und zu sichern.

Topics

- AWS verwaltete Richtlinien für Amazon Aurora DSQL
- Datenschutz in Amazon Aurora DSQL
- Datenverschlüsselung für Amazon Aurora DSQL
- Identitäts- und Zugriffsverwaltung für Aurora DSQL
- Ressourcenbasierte Richtlinien für Aurora DSQL
- Verwenden von serviceverknüpften Rollen in Aurora DSQL
- Verwenden von IAM-Bedingungsschlüsseln mit Amazon Aurora DSQL
- Vorfallreaktion in Amazon Aurora DSQL
- Compliance-Validierung für Amazon Aurora DSQL

- Ausfallsicherheit in Amazon Aurora DSQL
- Infrastruktursicherheit in Amazon Aurora DSQL
- Konfigurations- und Schwachstellenanalyse in Amazon Aurora DSQL
- Serviceübergreifende Confused-Deputy-Prävention
- Bewährte Methoden für die Aurora DSQL-Sicherheit

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 sind so konzipiert, dass sie Berechtigungen für viele gängige Anwendungsfälle bereitstellen, 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 für alle AWS Kunden verfügbar sind. 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 `AmazonAuroraDSQLFullAccess` an Ihre Benutzer, Gruppen und Rollen anfügen.

Diese Richtlinie gewährt administrative Berechtigungen, die vollständigen Administratorzugriff auf Aurora DSQL ermöglichen. Prinzipale mit diesen Berechtigungen können:

- Erstellen, Löschen und Aktualisieren von Aurora DSQL-Clustern und Clustern mit mehreren Regionen

- Cluster-Inline-Richtlinien verwalten (Richtlinien erstellen, anzeigen, aktualisieren und löschen)
- Entfernen und Hinzufügen von Tags bei einem Cluster
- Auflistung und Anzeige von Clustern und Informationen zu einzelnen Clustern
- Anzeige von an Aurora DSQL-Cluster angehängten Tags
- Datenbankverbindung als beliebiger Benutzer herstellen (einschließlich als Administrator)
- Durchführen von Sicherungs- und Wiederherstellungsvorgängen für Aurora DSQL-Cluster, einschließlich Starten, Stoppen und Überwachen von Sicherungs- und Wiederherstellungsaufträgen
- Verwenden Sie vom Kunden verwaltete AWS KMS Schlüssel für die Clusterverschlüsselung
- Sehen Sie sich alle Metriken CloudWatch für ihr Konto an
- Verwenden Sie AWS Fault Injection Service (AWS FIS), um Fehler in Aurora DSQL-Cluster für Fehlertoleranztests einzuschleusen
- Erstellen serviceverknüpfter Rollen für den `dsql.amazonaws.com`-Service, der für die Erstellung von Clustern erforderlich ist

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt Prinzipalen vollen Zugriff auf Aurora DSQL.
- `cloudwatch`— erteilt die Erlaubnis, metrische Datenpunkte auf Amazon CloudWatch zu veröffentlichen.
- `iam`— gewährt die Berechtigung zum Erstellen einer serviceverknüpften Rolle.
- `backup and restore`— gewährt Berechtigungen zum Starten, Stoppen und Überwachen von Sicherungs- und Wiederherstellungsaufträgen für Aurora DSQL-Cluster.
- `kms`— gewährt die erforderlichen Berechtigungen zum Validieren des Zugriffs auf kundenseitig verwaltete Schlüssel, die für die Aurora DSQL-Clusterverschlüsselung beim Erstellen, Aktualisieren oder Herstellen einer Verbindung zu Clustern verwendet werden.
- `fis`— gewährt Nutzungsberechtigungen AWS Fault Injection Service (AWS FIS) zum Einschleusen von Fehlern in Aurora DSQL-Cluster für Fehlertoleranztests.

Sie finden die von `AmazonAuroraDSQLFullAccess` Richtlinie in der IAM-Konsole und im Referenzleitfaden zur AWS -verwalteten Richtlinie.

AWS verwaltete Richtlinie: AmazonAurora DSQLRead OnlyAccess

Sie können `AmazonAuroraDSQLReadOnlyAccess` an Ihre Benutzer, Gruppen und Rollen anfügen.

Erlaubt den Lesezugriff auf Aurora DSQL. Prinzipale mit diesen Berechtigungen können Cluster auflisten und Informationen zu einzelnen Clustern einsehen. Sie können die an Aurora DSQL-Cluster angehängten Tags und die Cluster-Inline-Richtlinien einsehen. Sie können alle Metriken aus CloudWatch Ihrem Konto abrufen und einsehen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt Lesezugriff für alle Ressourcen in Aurora DSQL.
- `cloudwatch`— erteilt die Erlaubnis, stapelweise Mengen von CloudWatch metrischen Daten abzurufen und metrische Berechnungen mit abgerufenen Daten durchzuführen

Sie finden die `AmazonAuroraDSQLReadOnlyAccess`-Richtlinie in der IAM-Konsole und im Referenzleitfaden zur von AWS verwalteten Richtlinie.

AWS verwaltete Richtlinie: AmazonAurora DSQLConsole FullAccess

Sie können `AmazonAuroraDSQLConsoleFullAccess` an Ihre Benutzer, Gruppen und Rollen anfügen.

Ermöglicht vollen Administratorzugriff auf Amazon Aurora DSQL über die AWS-Managementkonsole. Prinzipale mit diesen Berechtigungen können:

- Aurora DSQL-Cluster, einschließlich Clustern mit mehreren Regionen erstellen, löschen und aktualisieren
- Cluster-Inline-Richtlinien über die Konsole verwalten (Richtlinien erstellen, anzeigen, aktualisieren und löschen)
- Auflistung und Anzeige von Clustern und Informationen zu einzelnen Clustern
- Sehen Sie sich Tags für jede Ressource in Ihrem Konto an

- Datenbankverbindung als beliebiger Benutzer herstellen (einschließlich als Administrator)
- Durchführen von Sicherungs- und Wiederherstellungsvorgängen für Aurora DSQL-Cluster, einschließlich Starten, Stoppen und Überwachen von Sicherungs- und Wiederherstellungsaufträgen
- Verwenden Sie vom Kunden verwaltete AWS KMS Schlüssel für die Clusterverschlüsselung
- Starten Sie AWS CloudShell von AWS-Managementkonsole
- Sehen Sie sich alle Metriken CloudWatch in Ihrem Konto an
- Verwenden Sie AWS Fault Injection Service (AWS FIS), um Fehler in Aurora DSQL-Cluster für Fehlertoleranztests einzuschleusen
- Erstellen serviceverknüpfter Rollen für den `dsql.amazonaws.com`-Service, der für die Erstellung von Clustern erforderlich ist

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

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `dsql`— gewährt volle Administratorrechte für alle Ressourcen in Aurora DSQL über AWS-Managementkonsole.
- `cloudwatch`— erteilt die Berechtigung zum Abrufen von Batchmengen von CloudWatch Metrikdaten und zum Durchführen von metrischen Berechnungen mit abgerufenen Daten.
- `tag`— erteilt die Erlaubnis, 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 zum Validieren des Zugriffs auf kundenseitig verwaltete Schlüssel, 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`— gewährt die Berechtigung, Amazon VPC-Endpunktinformationen einzusehen, die für Aurora DSQL-Verbindungen benötigt werden.

- `fis`— gewährt Berechtigungen AWS FIS zur Eingabe von Fehlern in Aurora DSQL-Cluster für Fehlertoleranztests.
- `access-analyzer:ValidatePolicy`erteilt die Erlaubnis für den Linter im Policy-Editor, der in Echtzeit Feedback zu Fehlern, Warnungen und Sicherheitsproblemen in der aktuellen Richtlinie liefert.
- `fis`— gewährt Nutzungsberechtigungen AWS Fault Injection Service (AWS FIS) zum Einschleusen von Fehlern in Aurora DSQL-Cluster für Fehlertoleranztests.

Sie finden die `AmazonAuroraDSQLConsoleFullAccess`-Richtlinie in der IAM-Konsole und im Referenzleitfaden zur von AWS verwalteten Richtlinie.

AWS verwaltete Richtlinie: Aurora DSQLService RolePolicy

Sie können Aurora nicht DSQLService RolePolicy an Ihre IAM-Entitäten anhängen. Diese Richtlinie ist an eine servicegebundene Rolle angehängt, mit der Aurora DSQL auf Kontoressourcen zugreifen kann.

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-Dokumentverlauf, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderungen	Beschreibung	Date
AmazonAuroraDSQLFullZugriff und Aktualisierung AmazonAurora DSQLConsole FullAccess	Unterstützung für die AWS Fault Injection Service (AWS FIS) -Integration mit Aurora DSQL hinzugefügt. Dadurch können Sie Ausfälle in Aurora	19. August 2025

Änderungen	Beschreibung	Date
	DSQL-Cluster mit einer oder mehreren Regionen einschleu sen, um die Fehlertoleranz Ihrer Anwendungen zu testen. Sie können in der AWS FIS Konsole Versuchsvorlagen erstellen, um Fehlerszenarien zu definieren und bestimmte Aurora DSQL-Cluster für Tests als Ziel zu verwenden. Weitere Informationen zu diesen Richtlinien finden Sie unter AmazonAuroraDSQLFu llZugriff und AmazonAur oraDSQLConsoleFullAccess.	

Änderungen	Beschreibung	Date
AmazonAuroraDSQLFullZugriff AmazonAurora DSQLReadOnlyAccess, und AmazonAurora DSQLConsoleFullAccess Aktualisierung	Unterstützung für ressourcenbasierte Richtlinien (RBP) mit neuen Berechtigungen hinzugefügt:`PutClusterPolicy` , und`GetClusterPolicy` . `DeleteClusterPolicy` Diese Berechtigungen ermöglichen die Verwaltung von Inline-Richtlinien, die an Aurora DSQL-Cluster angehängt sind, um eine differenzierte Zugriffskontrolle zu gewährleisten. Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugriff, und AmazonAuroraDSQLReadOnlyAccess. AmazonAuroraDSQLConsoleFullAccess	15. Oktober 2025

Änderungen	Beschreibung	Date
AmazonAuroraDSQLFullZugriffsupdate	Es wurde die Möglichkeit hinzugefügt, Backup- und Wiederherstellungsoperationen für Aurora DSQL-Cluster durchzuführen – einschließlich Starten, Stoppen und Überwachen von Aufträgen. Weiterhin wurde die Möglichkeit hinzugefügt, kundenseitig verwaltete KMS-Schlüssel für die Clusterverschlüsselung zu verwenden. Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugreifen und Verwenden von serviceverknüpften Rollen in Aurora DSQL.	21. Mai 2025

Änderungen	Beschreibung	Date
AmazonAuroraDSQLConsoleFullAccess update	Ermöglicht jetzt die Durchführung von Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster über die AWS Console Home. Dies beinhaltet das Starten, Stoppen und Überwachen von Aufträgen. Weiterhin wird jetzt auch die Verwendung von kundenseitig verwalteten KMS-Schlüsseln für die Clusterverschlüsselung und den AWS CloudShell-Start unterstützt. Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	21. Mai 2025

Änderungen	Beschreibung	Date
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 zur Steuerung, welche Cluster-Benutzer Sie ändern können. Die Richtlinie fügt auch die `GetVpcEndpointServiceName` -Berechtigung hinzu, die Ihnen hilft, eine Verbindung zu Ihren Aurora DSQL-Clustern über AWS PrivateLink herzustellen. Weitere Informationen finden Sie unter Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugreifen und Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025

Änderungen	Beschreibung	Date
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 servicerknüpften Rollen in Aurora DSQL.	13. Mai 2025

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

Änderungen	Beschreibung	Date
AuroraDsqlServiceLinkedRole Policy update	Es wurde die Möglichkeit hinzugefügt, Metriken in den `AWS/AuroraDSQL` - und `AWS/Usage CloudWatch` - Namespaces der Richtlinie zu veröffentlichen. Auf diese Weise kann der zugehörig e Dienst oder die zugehörig e Rolle umfassendere Nutzungs- und Leistungs daten an Ihre CloudWatch Umgebung senden. Weitere Informationen finden Sie unter AuroraDsq lServiceLinkedRolePolicyund Verwenden von serviceve rknüpften Rollen in Aurora DSQL.	8. Mai 2025
Seite wurde erstellt.	Mit der Verfolgung AWS verwalteter Richtlinien im Zusammenhang mit Amazon Aurora DSQL wurde begonnen	03. Dezember 2024

Datenschutz in Amazon Aurora DSQL

Das Modell der geteilten Verantwortung bezieht sich auf den Datenschutz. Wie in diesem Modell beschrieben, ist verantwortlich für den Schutz der globalen Infrastruktur, auf der AWS Cloud alle Systeme laufen. 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 Trails zur Erfassung von Aktivitäten finden Sie unter Arbeiten mit Trails 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 die E-Mail-Adressen Ihrer Kunden in Tags oder Freitextfeldern wie Name einzugeben. Dazu gehört auch, wenn Sie mit der Konsole, der API oder oder arbeiten oder AWS CLI anderweitig verwenden AWS SDKs. 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 sehr robuste Speicherinfrastruktur, die für geschäftskritische und primäre Datenspeicher entwickelt wurde. Daten werden redundant auf mehreren Geräten in verschiedenen Anlagen einer Aurora DSQL-Region gespeichert.

Verschlüsselung während der Übertragung

Die Verschlüsselung bei der Übertragung ist für Sie standardmäßig vorkonfiguriert. Aurora DSQL verwendet TLS zur Verschlüsselung des gesamten Datenverkehrs zwischen Ihrem SQL-Client und Aurora DSQL.

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

- Aurora DSQL stellt HTTPS-Endpunkte zur Verschlüsselung von Daten während der Übertragung bereit.
- Um die Integrität von API-Anforderungen 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.

Compliance mit FIPS

Aurora DSQL-Datenebenen-Endpunkte (Cluster-Endpunkte, die für Datenbankverbindungen verwendet werden) verwenden standardmäßig FIPS 140-2-validierte kryptografische Module. Für Clusterverbindungen sind keine separaten FIPS-Endpunkte erforderlich.

Für den Betrieb auf der Kontrollebene bietet Aurora DSQL spezielle FIPS-Endpunkte in unterstützten Regionen. Weitere Informationen zu FIPS-Endpunkten auf Kontrollebene finden Sie unter Aurora DSQL-Endpunkte und Kontingente in der. Allgemeine AWS-Referenz

Siehe Verschlüsselung im Ruhezustand in Aurora DSQL zum Thema Verschlüsselung im Ruhezustand.

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 Direct Connect Verbindung. Weitere Informationen finden Sie unter Was ist Direct Connect?

Sie erhalten Zugriff auf Aurora DSQL über das Netzwerk, indem Sie AWS-veröffentlichte 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.

Datenschutz in Witness-Regionen

Wenn Sie einen Cluster mit mehreren Regionen erstellen, ermöglicht eine Witness-Region die automatische Wiederherstellung nach einem Ausfall, indem sie an der synchronen Replikation verschlüsselter Transaktionen teilnimmt. Wenn ein Peer-Cluster nicht mehr verfügbar ist, steht die Witness-Region weiterhin für die Validierung und Verarbeitung von Datenbank-Schreibvorgängen zur Verfügung, sodass kein Verfügbarkeitsverlust entsteht.

Witness-Regionen schützen und sichern Ihre Daten mithilfe der folgendern Designmerkmale:

- Die Witness-Region empfängt und speichert ausschließlich verschlüsselte Transaktionsprotokolle. Sie wird niemals Ihre Verschlüsselungsschlüssel hosten, speichern oder weitergeben.
- Die Witness-Region dient lediglich zur Protokollierung von Schreibtransaktionen und Quorumfunktionen. Sie wurde speziell so entwickelt, dass sie Ihre Daten nicht lesen kann.
- Die Witness-Region arbeitet ohne Clusterverbindungsendpunkte oder Abfrageprozessoren. Dadurch wird jeglicher Zugriff auf die Benutzerdatenbank verhindert.

Weitere Informationen zu Witness-Regionen finden Sie unter Konfigurieren von Clustern mit mehreren Regionen.

Konfiguration von SSL/TLS Zertifikaten für Aurora DSQL-Verbindungen

Aurora DSQL verlangt von allen Verbindungen eine Transport Layer Security (TLS)-Verschlüsselung. Ihr Client-System muss der Amazon Root Certificate Authority (Amazon Root CA 1) vertrauen, um

sichere Verbindungen herzustellen. Dieses Zertifikat ist bei vielen Betriebssystemen vorinstalliert. Dieser Abschnitt enthält Anleitungen zur Überprüfung des vorinstallierten Amazon Root CA 1-Zertifikats bei 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.

Überprüfen vorinstallierter Zertifikate

Bei den meisten Betriebssystemen ist Amazon Root CA 1 bereits vorinstalliert. Um dies zu überprüfen, führen Sie die folgenden Schritte aus.

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, wird die folgende Ausgabe angezeigt:

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

macOS

1. Öffnen Sie die Spotlight-Suche (Befehlstaste + Leerzeichen)
2. Suchen Sie nach Keychain Access
3. Wählen Sie unter Systemschlüsselketten die Option Systemstammverzeichnis 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`. Alternativ können Sie den Schritten unter Eine Verbindung von Windows aus herstellen folgen, um mithilfe von SSL eine Verbindung zu Ihrem Cluster herzustellen.

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

Installieren von Zertifikaten

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.

Installieren eines Linux-Zertifikats

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

1. Laden Sie das Stammzertifikat herunter:

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

2. Kopieren Sie das Zertifikat in den Vertrauensspeicher ein.

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

3. Aktualisieren Sie den CA Trust Store:

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

4. Überprüfen der Installation:

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

Installieren des macOS-Zertifikats

Diese Schritte zur Installation des Zertifikats sind optional. Die Schritte unter Installieren eines Linux-Zertifikats funktionieren auch für macOS.

1. Das Stammzertifikat herunterladen:

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

2. Fügen Sie das Zertifikat zur Systemschlüsselkette hinzu:

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

3. Überprüfen der Installation:

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

Verbindung mit SSL/TLS Verifizierung herstellen

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 ist installiert
- AWS CLI mit den entsprechenden Anmeldeinformationen konfiguriert
- Informationen zum Aurora DSQL-Cluster-Endpunkt

Eine Verbindung von Linux aus herstellen

1. Generieren und Festlegen des Authentifizierungstokens:

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

2. Mithilfe von Systemzertifikaten eine Verbindung herstellen (falls vorinstalliert):

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

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

3. Alternativ mithilfe eines heruntergeladenen Zertifikats eine Verbindung herstellen:

```
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 Datenbankverbindung-Steuerungsfunktionen von PostgreSQL 17.

Eine Verbindung von macOS aus herstellen

1. Generieren und Festlegen des Authentifizierungstokens:

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

2. Mithilfe von Systemzertifikaten eine Verbindung herstellen (falls vorinstalliert):

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

3. Alternativ laden Sie das Stammzertifikat herunter und speichern 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. Eine Verbindung mithilfe von psql herstellen:

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

Eine Verbindung von Windows aus herstellen

Verwenden der Eingabeaufforderung

1. Erstellen des Authentifizierungstokens:

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

2. Legen Sie die Passwortumgebungsvariable fest:

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

3. Legen Sie die SSL-Konfiguration fest:

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

4. Verbindung mit der Datenbank herstellen:

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

Verwenden PowerShell

1. Generieren und Festlegen des Authentifizierungstokens:

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

2. Die SSL-Konfiguration festlegen:

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

3. Verbindung mit der Datenbank herstellen:

```
 "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. Diese Verschlüsselung verwendet AWS Key Management Service (AWS KMS) für erhöhte Sicherheit. Diese Funktionalität trägt zur Verringerung des Betriebsaufwands und der Komplexität bei, die mit dem Schutz sensibler Daten einhergeht. Verschlüsselung im Ruhezustand:

- Reduzierung des betrieblichen Aufwands, der durch den Schutz sensibler Daten entsteht
- Aufbau sicherheitsrelevanter Anwendungen, die strenge Verschlüsselungsvorschriften und gesetzliche Auflagen erfüllen
- Hinzufügung einer zusätzlichen Datenschutzebene, wodurch Ihre Daten immer in einem verschlüsselten Cluster gesichert sind
- Einhaltung aller Unternehmensrichtlinien, Branchen- oder behördliche Vorschriften sowie Compliance-Anforderungen

Mit Aurora DSQL können Sie sicherheitsrelevante Anwendungen erstellen, die eine strenge Einhaltung der Verschlüsselungsvorschriften und der gesetzlichen Bestimmungen garantieren. 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
- Verschlüsselung im Ruhezustand in Aurora DSQL
- Verwendung AWS KMS und Datenschlüssel mit Aurora DSQL
- Autorisieren der Verwendung Ihres AWS KMS key für Aurora DSQL
- Aurora DSQL Verschlüsselungskontext
- Überwachung der Aurora DSQL-Interaktion mit AWS KMS
- Erstellen eines verschlüsselten Aurora DSQL-Clusters
- Entfernen oder Aktualisieren eines Schlüssels für Ihren Aurora DSQL-Cluster
- Überlegungen zur Verschlüsselung mit Aurora DSQL

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 -zustände finden Sie unter AWS Key Management Service Konzepte im AWS Key Management Service Entwicklerhandbuch. Wenn Sie einen neuen Cluster erstellen, können Sie für Ihre Clusterverschlüsselung aus den folgenden KMS-Schlüsseltypen wählen:

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. Es ist kein Code- oder Anwendungswechsel nötig, um verschlüsselte Cluster zu verwenden oder zu verwalten. Alle Aurora DSQL-Abfragen funktionieren mit Ihren verschlüsselten Daten.

Kundenseitig verwalteter 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 Kundenseitig verwaltete Schlüssel und AWS-eigene Schlüssel im 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) zur Verschlüsselung Ihrer Daten im Ruhezustand. Diese Verschlüsselung schützt Ihre Daten vor unbefugtem Zugriff auf den zugrunde liegenden Speicher. AWS KMS verwaltet die Verschlüsselungsschlüssel für Ihre Cluster. Sie können die standardmäßigen AWS-eigene Schlüssel verwenden, oder aber Ihre eigenen AWS KMS -Kundenseitig verwaltete Schlüssel auswählen. Weitere Informationen zur Festlegung und Verwaltung von Schlüsseln für Ihre Aurora DSQL-Cluster finden Sie unter Erstellen eines verschlüsselten Aurora DSQL-Clusters und Entfernen oder Aktualisieren eines Schlüssels für Ihren Aurora DSQL-Cluster.

Themen

- AWS-eigene Schlüssel
- Kundenseitig verwaltete 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 für den Datenschutz erforderlich werden. Weitere Informationen zu AWS-eigene Schlüssel finden Sie AWS-eigene Schlüsselim AWS Key Management Service Entwicklerhandbuch.

Kundenseitig verwaltete Schlüssel

Kundenseitig verwaltete Schlüssel erstellen, besitzen und verwalten Sie über Ihr AWS-Konto. Sie haben die vollständige Kontrolle über diese KMS-Schlüssel, einschließlich ihrer Richtlinien, Verschlüsselungsmaterialien, Tags und Aliasse. Weitere Informationen finden Sie unter Kundenseitig verwaltete Schlüssel im AWS Key Management Service -Entwicklerhandbuch.

Wenn Sie einen kundenseitig 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 kundenseitig verwalteten Schlüssel deaktivieren, Ihren Schlüssel für die Löschung planen oder eine Richtlinie festgelegt 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 auf `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 auf `KMS_KEY_INACCESSIBLE` gesetzt, wodurch er nicht mehr für Lese- und Schreibvorgänge verfügbar ist. Andere Peers setzen den normalen Betrieb fort.

> **Note**
>
> Wenn Aurora DSQL nicht auf Ihren kundenseitig verwalteten Schlüssel zugreifen kann, ändert sich Ihr Cluster-Verschlüsselungsstatus auf `KMS_KEY_INACCESSIBLE`. Nachdem Sie den Schlüsselzugriff wiederhergestellt haben, erkennt der Dienst die Wiederherstellung automatisch innerhalb von 15 Minuten. Weitere Informationen finden Sie unter Clusterleerlauf.
>
> Wenn bei Clustern mit mehreren Regionen der Schlüsselzugriff über einen längeren Zeitraum verloren geht, hängt die Cluster-Wiederherstellungszeit 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, Ihre Verschlüsselungsstrategie im Voraus zu planen, bevor Sie Ihren Cluster in Aurora DSQL implementieren. Wenn Sie sensible oder vertrauliche Daten in Aurora DSQL speichern, sollten Sie erwägen, eine 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
- Clusterschlüssel mit Aurora DSQL verwenden
- Clusterschlüssel-Caching

Verwenden von AWS KMS key s mit Aurora DSQL

Die 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 kundenseitig verwalteten Schlüssel in Ihrem AWS-Konto verschlüsseln. Für jeden Cluster können Sie einen anderen KMS-Schlüssel auswählen, auch wenn dieser an einer multiregionalen Konfiguration beteiligt ist.

Den KMS-Schlüssel für ein Cluster wählen Sie beim Erstellen oder Aktualisieren des Clusters aus. Sie können den KMS-Schlüssel für ein Cluster jederzeit entweder in der Aurora DSQL-Konsole oder mithilfe der `UpdateCluster`-Operation ändern. Der Vorgang des Schlüsselwechsels ist nahtlos und erfolgt ohne Ausfallzeiten oder Servicebeeinträchtigung.

> **Important**
>
> Aurora DSQL unterstützt ausschließlich symmetrische KMS-Schlüssel. Ein asymmetrischer KMS-Schlüssel kann nicht zum Verschlüsseln Ihrer Aurora DSQL-Cluster verwendet werden.

Ein kundenseitig verwalteter Schlüssel bietet die folgenden Vorteile:

- Sie erstellen und verwalten den KMS-Schlüssel selbst, einschließlich der Einstellung der Schlüsselrichtlinien und IAM-Richtlinien, um den Zugriff auf den KMS-Schlüssel zu steuern. 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 kundenseitig verwalteten 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 die AWS KMS Ressourcen- oder Anforderungskontingente angerechnet. Für kundenseitig verwaltete Schlüssel fällt bei jedem API-Aufruf eine Gebühr an, und die AWS KMS -Kontingente gelten für diese Schlüssel.

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 für den Verschlüsselungsschlüssel verwendet. Aurora DSQL verwendet diesen Clusterschlüssel zum Schutz von Datenverschlüsselungsschlüsseln, die wiederum zum Verschlüsseln der Clusterdaten verwendet werden. Aurora DSQL generiert für jede zugrunde liegende Struktur in einem Cluster einen eindeutigen Datenverschlüsselungsschlüssel, wobei mehrere Clusterelemente durch denselben Schlüssel geschützt sein können.

Um den Clusterschlüssel zu entschlüsseln, sendet Aurora DSQL eine Anfrage an, AWS KMS wenn Sie zum ersten Mal auf einen verschlüsselten Cluster zugreifen. Um die Verfügbarkeit des Clusters sicherzustellen, überprüft Aurora DSQL regelmäßig den Entschlüsselungszugriff auf den KMS-Schlüssel – auch wenn Sie gerade 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, damit sie on-demand für die Entschlüsselung der Clusterdaten verfügbar sind.

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

Clusterschlüssel-Caching

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 einen kundenseitig verwalteten Schlüssel in Ihrem Konto zum Schutz Ihres Aurora DSQL-Clusters verwenden, müssen die Richtlinien dieses Schlüssels Aurora DSQL die Berechtigung erteilen, ihn in Ihrem Namen zu verwenden.

Sie haben die vollständige Kontrolle über die Richtlinien eines kundenseitig verwalteten Schlüssels. 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 kundenseitig verwalteten 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.

Als Minimum benötigt Aurora DSQL die folgenden Berechtigungen für einen kundenseitig 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. Falls die in der Richtlinienerklärung angegebenen Prinzipalen keine Berechtigung zur Nutzung von Aurora DSQL haben, schlägt der Aufruf fehl – selbst wenn er vom Aurora DSQL-Service stammt.
- Der Bedingungsschlüssel `kms:ViaService` erlaubt die Berechtigungen nur, wenn die Anforderung im Auftrag der in der Richtlinienanweisung aufgeführten Prinzipale von Aurora DSQL stammt. Diese Prinzipale können diese Operationen nicht direkt aufrufen.

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 Anfrage zur Verschlüsselung von Daten einbeziehen, 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 von AWS CloudTrail.

Der Verschlüsselungskontext kann außerdem 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`. Dieser Wert ist die ID 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-Servicekonto. 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, um die darunterliegenden Schlüssel in der Hierarchie entschlüsseln zu können. 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"
}
```


Erstellen eines verschlüsselten Aurora DSQL-Clusters

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

Console

Um einen verschlüsselten Cluster im zu erstellen AWS-Managementkonsole

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 die Option Cluster aus.
3. Klicken Sie oben rechts auf Cluster erstellen und wählen Sie dann Einzelne Region aus.
4. Wählen Sie in den Cluster-Verschlüsselungseinstellungen eine der folgenden Optionen aus.
   - Akzeptieren Sie die Standardeinstellungen für die Verschlüsselung ohne zusätzliche AWS-eigener Schlüssel Kosten.
   - Wählen Sie Verschlüsselungseinstellungen anpassen (erweitert) aus, 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, um einen neuen Schlüssel in der AWS KMS Konsole zu erstellen.
5. Wählen Sie Cluster erstellen.

Um den Verschlüsselungstyp Ihres Clusters zu bestätigen, navigieren Sie zur Seite Cluster und wählen Sie dort die ID des Clusters aus, um dessen 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 einen eigenen Schlüssel besitzen und verwalten möchten, stellen Sie sicher, dass Sie die KMS-Schlüsselrichtlinie entsprechend festlegen. Beispiele und weitere Informationen finden Sie unter the section called “Schlüsselrichtlinie für einen kundenseitig verwalteten Schlüssel”.

CLI

Um einen Cluster zu erstellen, 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 dargestellt, ist die Verschlüsselung für den Cluster standardmäßig aktiviert, und der voreingestellte Verschlüsselungstyp ist ein von AWS verwalteter Schlüssel. Der Cluster ist nun mit dem standardmäßigen AWS-Schlüssel im Aurora DSQL-Servicekonto verschlüsselt.

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

So erstellen Sie einen mit Ihrem kundenseitig verwalteten Schlüssel verschlüsselten Cluster

- Verwenden Sie den folgenden Befehl, um einen Aurora DSQL-Cluster zu erstellen, und ersetzen Sie die rot markierte Schlüssel-ID durch die ID Ihres kundenseitig verwalteten Schlüssels.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Wie in den folgenden Verschlüsselungsdetails dargestellt, ist die Verschlüsselung für den Cluster standardmäßig aktiviert, und der Verschlüsselungstyp ist ein kundenseitig verwalteter KMS-Schlüssel. Der Cluster ist nun mit Ihrem Schlüssel verschlüsselt.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Entfernen oder Aktualisieren eines Schlüssels für Ihren Aurora DSQL-Cluster

Sie können das AWS-Managementkonsole 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. Im Folgenden sehen Sie, wie Sie die Verschlüsselungsschlüssel eines bestehenden Clusters über die Aurora DSQL-Konsole oder die AWS CLI aktualisieren.

Console

Um einen Verschlüsselungsschlüssel zu aktualisieren oder zu entfernen in AWS-Managementkonsole

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 die Option Cluster aus.
3. Suchen Sie in der Listenansicht die Zeile des Clusters, den Sie aktualisieren möchten, und wählen Sie sie aus.
4. Klicken Sie auf das Menü Aktionen und dann auf Bearbeiten.
5. Wählen Sie in den Cluster-Verschlüsselungseinstellungen eine der folgenden Optionen aus, 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), falls sie nicht bereits ausgewählt ist. Suchen Sie dann nach der ID oder dem Alias des Schlüssels, den Sie verwenden möchten, und wählen Sie dies 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 mit AWS CLI dem einen verschlüsselten Cluster aktualisieren können.

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` in der Clusterbeschreibung ist auf `ENABLED` gesetzt, 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.

So aktualisieren Sie einen verschlüsselten Cluster mit einem kundenseitig verwalteten Schlüssel für Aurora DSQL

Aktualisieren Sie den verschlüsselten Cluster wie im folgenden Beispiel gezeigt:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

Der `EncryptionStatus` in der Clusterbeschreibung wechselt zu `UPDATING`, und der `EncryptionType` ist `CUSTOMER_MANAGED_KMS_KEY`. Nachdem Aurora DSQL den neuen

Schlüssel vollständig in der Plattform propagiert hat, wird der Verschlüsselungsstatus auf `ENABLED` gesetzt.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

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

Überlegungen zur Verschlüsselung mit Aurora DSQL

- Aurora DSQL verschlüsselt alle Clusterdaten im Ruhezustand. Sie können diese Verschlüsselung nicht deaktivieren oder nur einzelne 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 Datenschutzzustände 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 `encryptionStatus` des Clusters auf `ENABLED` gesetzt ist und Sie die `kmsKeyArn` des neuen kundenseitig verwalteten Schlüssels sehen.

- Wenn Sie Ihren kundenseitig verwalteten Schlüssel deaktivieren oder den Zugriff für Aurora DSQL auf Ihren Schlüssel widerrufen, wechselt Ihr Cluster in den Status `IDLE`.
- Die DSQL-API AWS-Managementkonsole 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 kundenseitig 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-Managementkonsole AWS CLI, oder der Amazon Aurora SQL API vor.

Identitäts- und Zugriffsverwaltung für Aurora DSQL

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren steuern, wer für die Nutzung von Aurora DSQL-Ressourcen authentifiziert (angemeldet) und autorisiert (mit Berechtigungen versehen) werden kann. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Funktionsweise von Amazon Aurora DSQL mit IAM
- Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL
- Fehlerbehebung für Amazon Aurora DSQL-Identität und -Zugriff

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung für Amazon Aurora DSQL-Identität und -Zugriff).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe Funktionsweise von Amazon Aurora DSQL mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe 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 sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen

aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten 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 Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen 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. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.

- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Funktionsweise von Amazon Aurora DSQL mit IAM

Bevor Sie IAM für die Aurora DSQL-Zugriffsverwaltung einsetzen, informieren Sie sich bitte hier, welche IAM-Features Sie mit Aurora DSQL verwenden können.

IAM-Features, die Sie mit Amazon Aurora DSQL verwenden können

IAM-Feature	Aurora DSQL-Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Ja
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Bedingungsschlüssel für die Richtlinie	Ja
ACLs	Nein

IAM-Feature	Aurora DSQL-Unterstützung
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 Aurora DSQL-Richtlinien

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. 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 Aurora DSQL-Richtlinien

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: Ja

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. Prinzipale können Konten, Benutzer, Rollen, verbundene Benutzer oder AWS-Services umfassen. Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS-verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Informationen zum Erstellen und Verwalten ressourcenbasierter Richtlinien für Aurora DSQL-Cluster finden Sie unter Ressourcenbasierte Richtlinien für Aurora DSQL.

Richtlinienaktionen für Aurora DSQL

Unterstützt Richtlinienaktionen: Ja

Administratoren können AWS JSON-Richtlinien verwenden, um festzulegen, 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. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der Aurora DSQL-Aktionen finden Sie unter Von Amazon Aurora DSQL definierte Aktionen in der Service Authorization-Referenz.

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.

Richtlinienressourcen 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. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, 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 zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen 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.

Richtlinien-Bedingungsschlüssel für Aurora DSQL

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: 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 `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. 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. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste von Aurora DSQL-Bedingungsschlüsseln finden Sie unter Bedingungsschlüssel für Amazon Aurora DSQL in der Service-Authorization-Referenz. Informationen dazu, für welche Aktionen und Ressourcen 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, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS -Ressourcen anhängen und dann ABAC-Richtlinien so entwerfen, dass Operationen möglich sind, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

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.

Verwenden temporärer Anmeldeinformationen mit Aurora DSQL

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen den kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Anmeldeinformationen in IAM und AWS-Services , die mit IAM funktionieren im IAM-Benutzerhandbuch.

Serviceübergreifende Prinzipal-Berechtigungen für Aurora DSQL

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen AWS-Service an nachgeschaltete Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anforderungen 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**
>
> Eine Änderung der Berechtigungen für eine Servicerolle könnte die Aurora DSQL-Funktionalität beeinträchtigen. Bearbeiten Sie Servicerollen nur, wenn Aurora DSQL eine Anleitung dazu bereitstellt.

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.

Details zum Erstellen oder Verwalten von serviceverknüpften Aurora DSQL-Rollen finden Sie unter Verwenden von serviceverknüpften Rollen in Aurora DSQL.

Beispiele für identitätsbasierte Richtlinien für Amazon Aurora DSQL

Benutzer und Rollen haben standardmäßig nicht die Berechtigung, Aurora DSQL-Ressourcen zu erstellen oder zu ändern. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

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

- Best Practices für Richtlinien
- Verwenden der Aurora DSQL-Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Clusterverwaltung und Datenbankverbindung zulassen
- Aurora DSQL-Ressourcenzugriff basierend auf Tags

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand Aurora DSQL-Ressourcen in Ihrem Konto erstellen, auf sie zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen

AWS-Konto. Wenn Sie identitätsbasierte Richtlinien erstellen oder bearbeiten, befolgen Sie diese Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — 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 Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen 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. 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 Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der Aurora DSQL-Konsole

Sie müssen über einen Mindestsatz von Berechtigungen verfügen, um auf die Amazon Aurora DSQL-Konsole zugreifen zu können. 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": "*"
        }
    ]
}
```

Clusterverwaltung und Datenbankverbindung zulassen

Die folgende Richtlinie gewährt einem IAM-Benutzer die Berechtigung, einen bestimmten Aurora DSQL-Cluster zu verwalten und eine Verbindung zu diesem herzustellen. Die Richtlinie erstreckt sich auf Clusterverwaltung und Verbindungsaktionen auf einen einzelnen Cluster mit Amazon Resource Name (ARN), lässt aber alle Ressourcen zu, da diese Aktion keine Berechtigungen `dsql:ListClusters` auf Ressourcenebene unterstützt.

In diesem Beispiel wird eine `dsql:DbConnectAdmin` Verbindung mit der Rolle hergestellt. `admin` Um stattdessen eine Verbindung mit einer benutzerdefinierten Datenbankrolle herzustellen, `dsql:DbConnectAdmin` ersetzen Sie durch`dsql:DbConnect`. Weitere Informationen finden Sie unter Authentifizierung und Autorisierung für Aurora DSQL.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowClusterManagement",
            "Effect": "Allow",
            "Action": [
                "dsql:GetCluster",
                "dsql:UpdateCluster",
                "dsql:DeleteCluster",
                "dsql:DbConnectAdmin",
                "dsql:TagResource",
                "dsql:ListTagsForResource",
                "dsql:UntagResource"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/my-cluster-id"
        },
        {
            "Sid": "AllowListClusters",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        }
    ]
}
```

Aurora DSQL-Ressourcenzugriff basierend auf Tags

Sie können Bedingungen in Ihrer identitätsbasierten Richtlinie verwenden, um den Zugriff auf Aurora DSQL-Ressourcen anhand von Tags zu steuern. Das folgende Beispiel zeigt, wie Sie eine Richtlinie erstellen können, die die Anzeige eines Clusters ermöglicht. Die Richtlinie gewährt jedoch nur dann Berechtigungen, wenn das Cluster-Tag den Wert des Benutzernamens dieses Benutzers `Owner` hat. Diese Richtlinie gewährt auch die Berechtigungen, die für die Ausführung dieser Aktion auf der Konsole erforderlich sind.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListClustersInConsole",
```

```
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        },
        {
            "Sid": "ViewClusterIfOwner",
            "Effect": "Allow",
            "Action": "dsql:GetCluster",
            "Resource": "arn:aws:dsql:*:*:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

Sie können diese Richtlinie den IAM-Benutzern in Ihrem Konto anfügen. Wenn ein benannter Benutzer `richard-roe` versucht, einen Aurora DSQL-Cluster aufzurufen, muss der Cluster mit `0wner=richard-roe` oder `owner=richard-roe` gekennzeichnet werden. Andernfalls verweigert IAM den Zugriff. Der Tag-Schlüssel `0wner` der Bedingung stimmt sowohl mit `0wner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

Die folgende Richtlinie ermöglicht es einem Benutzer, Cluster nur zu erstellen, wenn er den Cluster mit seinem eigenen Benutzernamen als kennzeichnet. `0wner` Außerdem erlaubt sie das Tagging nur für Cluster, die der Benutzer bereits besitzt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedCluster",
            "Effect": "Allow",
            "Action": "dsql:CreateCluster",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        },
        {
            "Sid": "AllowTagOwnedClusters",
            "Effect": "Allow",
            "Action": "dsql:TagResource",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

Fehlerbehebung für Amazon Aurora DSQL-Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die beim Arbeiten mit Aurora DSQL und IAM auftreten könnten.

Themen

- Ich bin nicht autorisiert, eine Aktion in Aurora DSQL auszuführen.
- Ich bin nicht berechtigt, iam auszuführen: PassRole
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine Aurora DSQL-Ressourcen ermöglichen

Ich bin nicht autorisiert, eine Aktion in Aurora DSQL auszuführen.

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 ein `mateojackson` versucht, über die Konsole Details zu der *`my-dsql-cluster`*-Ressource anzuzeigen, jedoch nicht über die entsprechenden *`GetCluster`*-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, eine `iam:PassRole`-Aktion auszuführen, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an Aurora DSQL übergeben zu 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, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen 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 dazu, ob Aurora DSQL diese Features unterstützt, finden Sie unter Funktionsweise von Amazon Aurora DSQL mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen in AWS-Konten Ihrem Besitz gewähren können, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , dem Sie gehören.
- 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.

Ressourcenbasierte Richtlinien für Aurora DSQL

Verwenden Sie ressourcenbasierte Richtlinien für Aurora DSQL, um den Zugriff auf Ihre Cluster mithilfe von JSON-Richtliniendokumenten einzuschränken oder zu gewähren, die direkt mit Ihren Clusterressourcen verknüpft sind. Diese Richtlinien bieten eine detaillierte Kontrolle darüber, wer unter welchen Bedingungen auf Ihren Cluster zugreifen kann.

Aurora DSQL-Cluster sind standardmäßig über das öffentliche Internet zugänglich, wobei die IAM-Authentifizierung die primäre Sicherheitskontrolle ist. Mithilfe ressourcenbasierter Richtlinien können Sie Zugriffsbeschränkungen hinzufügen, insbesondere um den Zugriff über das öffentliche Internet zu blockieren.

Ressourcenbasierte Richtlinien arbeiten mit identitätsbasierten IAM-Richtlinien zusammen. AWS bewertet beide Arten von Richtlinien, um die endgültigen Berechtigungen für jede Zugriffsanfrage auf Ihren Cluster zu ermitteln. Standardmäßig sind Aurora DSQL-Cluster innerhalb eines Kontos zugänglich. Wenn ein IAM-Benutzer oder eine IAM-Rolle über Aurora DSQL-Berechtigungen verfügt, kann er auf Cluster zugreifen, ohne dass eine ressourcenbasierte Richtlinie angehängt ist.

> **Note**
>
> Änderungen an ressourcenbasierten Richtlinien sind letztlich konsistent und werden in der Regel innerhalb einer Minute wirksam.

Weitere Informationen zu den Unterschieden zwischen identitätsbasierten und ressourcenbasierten Richtlinien finden Sie unter Identitätsbasierte Richtlinien und ressourcenbasierte Richtlinien im IAM-Benutzerhandbuch.

Wann sollten ressourcenbasierte Richtlinien verwendet werden

Ressourcenbasierte Richtlinien sind in diesen Szenarien besonders nützlich:

- Netzwerkbasierte Zugriffskontrolle — Beschränken Sie den Zugriff auf der Grundlage der VPC oder IP-Adresse, von der Anfragen stammen, oder blockieren Sie den öffentlichen Internetzugang vollständig. Verwenden Sie Bedingungsschlüssel wie `aws:SourceVpc` und`aws:SourceIp`, um den Netzwerkzugriff zu kontrollieren.
- Mehrere Teams oder Anwendungen — Gewähren Sie mehreren Teams oder Anwendungen Zugriff auf denselben Cluster. Anstatt einzelne IAM-Richtlinien für jeden Prinzipal zu verwalten, definieren Sie Zugriffsregeln nur einmal im Cluster.
- Komplexer bedingter Zugriff — Steuern Sie den Zugriff auf der Grundlage mehrerer Faktoren wie Netzwerkattributen, Anforderungskontext und Benutzerattributen. Sie können mehrere Bedingungen in einer einzigen Richtlinie kombinieren.
- Zentralisierte Sicherheitssteuerung — Ermöglichen Sie es Cluster-Besitzern, den Zugriff mithilfe einer vertrauten AWS Richtliniensyntax zu kontrollieren, die sich in Ihre bestehenden Sicherheitspraktiken einfügt.

> **Note**
>
> Kontoübergreifender Zugriff wird für ressourcenbasierte Aurora DSQL-Richtlinien noch nicht unterstützt, wird aber in future Versionen verfügbar sein.

Wenn jemand versucht, eine Verbindung zu Ihrem Aurora DSQL-Cluster herzustellen, AWS bewertet Ihre ressourcenbasierte Richtlinie als Teil des Autorisierungskontextes zusammen mit allen relevanten IAM-Richtlinien, um festzustellen, ob die Anfrage zugelassen oder abgelehnt werden soll.

Ressourcenbasierte Richtlinien können Prinzipalen Zugriff gewähren, die sich innerhalb desselben Kontos wie der Cluster befinden. AWS Bei Clustern mit mehreren Regionen hat jeder regionale Cluster seine eigene ressourcenbasierte Richtlinie, die bei Bedarf regionsspezifische Zugriffskontrollen ermöglicht.

> **Note**
>
> Bedingungskontextschlüssel können je nach Region variieren (z. B. VPC IDs).

Topics

- Cluster mit ressourcenbasierten Richtlinien erstellen
- Hinzufügen und Bearbeiten ressourcenbasierter Richtlinien für Cluster
- Ressourcenbasierte Richtlinien anzeigen
- Entfernen ressourcenbasierter Richtlinien
- Allgemeine Beispiele für ressourcenbasierte Richtlinien
- Blockieren des öffentlichen Zugriffs mit ressourcenbasierten Richtlinien in Aurora DSQL
- Aurora DSQL API-Operationen und ressourcenbasierte Richtlinien

Cluster mit ressourcenbasierten Richtlinien erstellen

Sie können beim Erstellen eines neuen Clusters ressourcenbasierte Richtlinien anhängen, um sicherzustellen, dass die Zugriffskontrollen von Anfang an vorhanden sind. Jedem Cluster kann eine einzelne Inline-Richtlinie direkt an den Cluster angehängt werden.

AWS Management-Konsole

Um bei der Clustererstellung eine ressourcenbasierte Richtlinie hinzuzufügen

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 Cluster erstellen.
3. Konfigurieren Sie Ihren Clusternamen, Ihre Tags und Einstellungen für mehrere Regionen nach Bedarf.
4. Suchen Sie im Abschnitt Clustereinstellungen nach der Option Ressourcenbasierte Richtlinie.
5. Aktivieren Sie die Option Ressourcenbasierte Richtlinie hinzufügen.
6. Geben Sie Ihr Richtliniendokument im JSON-Editor ein. Um beispielsweise den öffentlichen Internetzugang zu blockieren:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Sie können die Option Aussage bearbeiten oder Neue Erklärung hinzufügen verwenden, um Ihre Richtlinie zu erstellen.
8. Schließen Sie die verbleibende Clusterkonfiguration ab und wählen Sie Create cluster aus.

AWS CLI

Verwenden Sie den `--policy` Parameter beim Erstellen eines Clusters, um eine Inline-Richtlinie anzuhängen:

```
aws dsql create-cluster --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}'
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}

response = client.create_cluster(
    policy=json.dumps(policy)
)
```

```
print(f"Cluster created: {response['identifier']}")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
    }
  }]
}
""";

CreateClusterRequest request = CreateClusterRequest.builder()
    .policy(policy)
    .build();

CreateClusterResponse response = client.createCluster(request);
System.out.println("Cluster created: " + response.identifier());
```

Hinzufügen und Bearbeiten ressourcenbasierter Richtlinien für Cluster

AWS Management-Konsole

Um einem vorhandenen Cluster eine ressourcenbasierte Richtlinie hinzuzufügen

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 Ihren Cluster aus der Cluster-Liste aus, um die Cluster-Detailseite zu öffnen.
3. Wählen Sie die Registerkarte Berechtigungen.
4. Wählen Sie im Abschnitt Ressourcenbasierte Richtlinie die Option Richtlinie hinzufügen aus.
5. Geben Sie Ihr Richtliniendokument im JSON-Editor ein. Sie können die Option Erklärung bearbeiten oder Neue Erklärung hinzufügen verwenden, um Ihre Richtlinie zu erstellen.
6. Wählen Sie Richtlinie hinzufügen aus.

Um eine bestehende ressourcenbasierte Richtlinie zu bearbeiten

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 Ihren Cluster aus der Cluster-Liste aus, um die Cluster-Detailseite zu öffnen.
3. Wählen Sie die Registerkarte Berechtigungen.
4. Wählen Sie im Abschnitt Ressourcenbasierte Richtlinie die Option Bearbeiten aus.
5. Ändern Sie das Richtliniendokument im JSON-Editor. Sie können die Option Erklärung bearbeiten oder Neue Erklärung hinzufügen verwenden, um Ihre Richtlinie zu aktualisieren.
6. Wählen Sie Änderungen speichern aus.

AWS CLI

Verwenden Sie den `put-cluster-policy` Befehl, um eine neue Richtlinie anzuhängen oder eine bestehende Richtlinie auf einem Cluster zu aktualisieren:

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": {"aws:SourceVpc": "true"}
        }
    }]
}

response = client.put_cluster_policy(
    identifier='your_cluster_id',
    policy=json.dumps(policy)
)
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.PutClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
```

```
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "Null": {"aws:SourceVpc": "true"}
    }
  }]
}
""";

PutClusterPolicyRequest request = PutClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .policy(policy)
    .build();

client.putClusterPolicy(request);
```

Ressourcenbasierte Richtlinien anzeigen

Sie können sich die ressourcenbasierten Richtlinien ansehen, die Ihren Clustern zugeordnet sind, um sich über die aktuellen Zugriffskontrollen zu informieren.

AWS Management-Konsole

Um ressourcenbasierte Richtlinien anzuzeigen

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 Ihren Cluster aus der Cluster-Liste aus, um die Cluster-Detailseite zu öffnen.
3. Wählen Sie die Registerkarte Berechtigungen.
4. Sehen Sie sich die beigefügte Richtlinie im Abschnitt Ressourcenbasierte Richtlinie an.

AWS CLI

Verwenden Sie den `get-cluster-policy` Befehl, um die ressourcenbasierte Richtlinie eines Clusters anzuzeigen:

```
aws dsql get-cluster-policy --identifier your_cluster_id
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

response = client.get_cluster_policy(
    identifier='your_cluster_id'
)

# Parse and pretty-print the policy
policy = json.loads(response['policy'])
print(json.dumps(policy, indent=2))
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyRequest;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyResponse;

DsqlClient client = DsqlClient.create();

GetClusterPolicyRequest request = GetClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

GetClusterPolicyResponse response = client.getClusterPolicy(request);
System.out.println("Policy: " + response.policy());
```

Entfernen ressourcenbasierter Richtlinien

Sie können ressourcenbasierte Richtlinien aus Clustern entfernen, um die Zugriffskontrollen zu ändern.

> **Important**
>
> Wenn Sie alle ressourcenbasierten Richtlinien aus einem Cluster entfernen, wird der Zugriff vollständig durch identitätsbasierte IAM-Richtlinien gesteuert.

AWS Management-Konsole

Um eine ressourcenbasierte Richtlinie zu entfernen

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 Ihren Cluster aus der Cluster-Liste aus, um die Cluster-Detailseite zu öffnen.
3. Wählen Sie die Registerkarte Berechtigungen.
4. Wählen Sie im Abschnitt Ressourcenbasierte Richtlinie die Option Löschen aus.
5. Geben Sie im Bestätigungsdialogfeld ein, **`confirm`** um den Löschvorgang zu bestätigen.
6. Wählen Sie Löschen aus.

AWS CLI

Verwenden Sie den `delete-cluster-policy` Befehl, um eine Richtlinie aus einem Cluster zu entfernen:

```
aws dsql delete-cluster-policy --identifier your_cluster_id
```

AWS SDKs

Python

```
import boto3

client = boto3.client('dsql')

response = client.delete_cluster_policy(
    identifier='your_cluster_id'
)

print("Policy deleted successfully")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

DeleteClusterPolicyRequest request = DeleteClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

client.deleteClusterPolicy(request);
System.out.println("Policy deleted successfully");
```

Allgemeine Beispiele für ressourcenbasierte Richtlinien

Diese Beispiele zeigen gängige Muster für die Steuerung des Zugriffs auf Ihre Aurora DSQL-Cluster. Sie können diese Muster kombinieren und ändern, um Ihre spezifischen Zugriffsanforderungen zu erfüllen.

Sperren Sie den öffentlichen Internetzugang

Diese Richtlinie blockiert Verbindungen zu Ihren Aurora DSQL-Clustern aus dem öffentlichen Internet (ohne VPC). Die Richtlinie legt nicht fest, von welcher VPC Kunden eine Verbindung herstellen können, sondern nur, dass sie sich von einer VPC aus verbinden müssen. Um den Zugriff auf eine bestimmte VPC zu beschränken, verwenden Sie ihn `aws:SourceVpc` zusammen mit dem `StringEquals` Bedingungsoperator.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
```

```
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> In diesem Beispiel wird nur `aws:SourceVpc` nach VPC-Verbindungen gesucht. Die Schlüssel `aws:VpcSourceIp` und `aws:SourceVpce` Condition bieten zusätzliche Granularität, sind aber für die grundlegende reine VPC-Zugriffskontrolle nicht erforderlich.

Verwenden Sie stattdessen diese Richtlinie, um eine Ausnahme für bestimmte Rollen bereitzustellen:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromOutsideVPC",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        },
        "StringNotEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::123456789012:role/ExceptionRole",
            "arn:aws:iam::123456789012:role/AnotherExceptionRole"
          ]
        }
```

```
      }
    }
  ]
}
```

Beschränken Sie den Zugriff auf die AWS Organisation

Diese Richtlinie schränkt den Zugriff auf Prinzipale innerhalb einer Organisation ein AWS :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "o-exampleorgid"
        }
      }
    }
  ]
}
```

Beschränken Sie den Zugriff auf eine bestimmte Organisationseinheit

Diese Richtlinie schränkt den Zugriff auf Prinzipale innerhalb einer bestimmten Organisationseinheit (OU) in einer AWS Organisation ein und bietet so eine detailliertere Kontrolle als der organisationsweite Zugriff:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotLike": {
          "aws:PrincipalOrgPaths": "o-exampleorgid/r-examplerootid/ou-exampleouid/*"
        }
      }
    }
  ]
}
```

Clusterrichtlinien für mehrere Regionen

Bei Clustern mit mehreren Regionen unterhält jeder regionale Cluster seine eigene Ressourcenpolitik, die regionsspezifische Kontrollen ermöglicht. Hier ist ein Beispiel mit unterschiedlichen Richtlinien pro Region:

US-Ost-1-Politik:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
        "Null": {
```

```
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

US-Ost-2-Politik:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> Die Schlüssel für den Bedingungskontext können zwischen diesen variieren AWS-Regionen (z. B. VPC IDs).

Blockieren des öffentlichen Zugriffs mit ressourcenbasierten Richtlinien in Aurora DSQL

Block Public Access (BPA) ist eine Funktion, die das Anhängen von ressourcenbasierten Richtlinien, die öffentlichen Zugriff auf Ihre Aurora DSQL-Cluster über Ihre Konten gewähren, identifiziert und

verhindert. AWS Mit BPA können Sie den öffentlichen Zugriff auf Ihre Aurora DSQL-Ressourcen verhindern. BPA führt während der Erstellung oder Änderung einer ressourcenbasierten Richtlinie Prüfungen durch und hilft Ihnen, Ihre Sicherheitslage mit Aurora DSQL zu verbessern.

BPA analysiert mittels Automated Reasoning den durch Ihre ressourcenbasierte Richtlinie gewährten Zugriff und warnt Sie, wenn solche Berechtigungen zum Zeitpunkt der Verwaltung einer ressourcenbasierten Richtlinie gefunden werden. Bei der Analyse wird der Zugriff über alle ressourcenbasierten Richtlinienanweisungen, Aktionen und die in Ihren Richtlinien verwendeten Bedingungsschlüssel überprüft.

> **Important**
>
> BPA trägt zum Schutz Ihrer Ressourcen bei, indem verhindert wird, dass öffentlicher Zugriff über die ressourcenbasierten Richtlinien gewährt wird, die direkt mit Ihren Aurora DSQL-Ressourcen wie Clustern verknüpft sind. Überprüfen Sie zusätzlich zur Aktivierung von BPA sorgfältig die folgenden Richtlinien, um sicherzustellen, dass sie keinen öffentlichen Zugriff gewähren:
>
> - Identitätsbasierte Richtlinien, die mit zugehörigen AWS Principals verknüpft sind (z. B. IAM-Rollen)
> - Ressourcenbasierte Richtlinien, die mit zugehörigen AWS Ressourcen verknüpft sind (z. B. AWS Key Management Service (KMS) -Schlüssel)

Sie müssen sicherstellen, dass der Prinzipal keinen *-Eintrag enthält oder dass keiner der angegebenen Bedingungsschlüssel den Zugriff der Prinzipale auf die Ressource einschränkt. Wenn die ressourcenbasierte Richtlinie AWS kontenübergreifend öffentlichen Zugriff auf Ihren Cluster gewährt, blockiert Aurora DSQL Sie daran, die Richtlinie zu erstellen oder zu ändern, bis die Spezifikation in der Richtlinie korrigiert und als nicht öffentlich eingestuft wurde.

Sie können eine Richtlinie als nicht öffentlich festlegen, indem Sie einen oder mehrere Prinzipale im `Principal`-Block angeben. Im folgenden Beispiel für eine ressourcenbasierte Richtlinie wird der öffentliche Zugriff blockiert, indem zwei Prinzipale angegeben werden.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
```

```
      "111122223333"
    ]
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Richtlinien, die den Zugriff durch die Angabe bestimmter Bedingungsschlüssel einschränken, gelten ebenfalls nicht als öffentlich. Neben der Auswertung des in der ressourcenbasierten Richtlinie angegebenen Prinzipals werden die folgenden vertrauenswürdigen Bedingungsschlüssel verwendet, um die Auswertung einer ressourcenbasierten Richtlinie für den nicht öffentlichen Zugriff abzuschließen:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

Damit eine ressourcenbasierte Richtlinie nicht öffentlich ist, dürfen die Werte für den Amazon-Ressourcennamen (ARN) und Zeichenfolgenschlüssel außerdem keine Platzhalter oder Variablen enthalten. Wenn Ihre ressourcenbasierte Richtlinie den `aws:PrincipalIsAWSService`-Schlüssel verwendet, müssen Sie sicherstellen, dass Sie den Schlüsselwert auf „true“ gesetzt haben.

Die folgende Richtlinie grenzt den Zugriff auf den Benutzer `Ben` im angegebenen Konto ein. Aufgrund dieser Bedingung ist der `Principal` eingeschränkt und wird als nicht öffentlich betrachtet.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

Das folgende Beispiel für eine nicht öffentliche ressourcenbasierte Richtlinie schränkt `sourceVPC` auf die Verwendung des `StringEquals`-Operators ein.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Aurora DSQL API-Operationen und ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien in Aurora DSQL steuern den Zugriff auf bestimmte API-Operationen. In den folgenden Abschnitten sind alle Aurora DSQL-API-Operationen nach Kategorien geordnet aufgeführt, mit Angabe, welche Operationen ressourcenbasierte Richtlinien unterstützen.

Die Spalte Unterstützt RBP gibt an, ob der API-Vorgang einer ressourcenbasierten Richtlinienbewertung unterzogen wird, wenn eine Richtlinie an den Cluster angehängt wird.

Tag APIs

API-Operation	Description	Unterstützt RBP
ListTagsForResource	Listet die Tags für eine Aurora DSQL-Ressource auf	Ja
TagResource	Fügt einer Aurora DSQL-Ressource Tags hinzu	Ja
UntagResource	Entfernt Tags aus einer Aurora DSQL-Ressource	Ja

Cluster-Verwaltung APIs

API-Operation	Description	Unterstützt RBP
CreateCluster	Erstellt einen Cluster	Nein
DeleteCluster	Löscht einen Cluster	Ja
GetCluster	Ruft Informationen über einen Cluster ab	Ja
GetVpcEndpointServiceName	Ruft den VPC-Endpunktdienstnamen für einen Cluster ab	Ja
ListClusters	Listet Cluster in Ihrem Konto auf	Nein
UpdateCluster	Aktualisiert die Konfiguration eines Clusters	Ja

Eigenschaft für mehrere Regionen APIs

API-Operation	Description	Unterstützt RBP
AddPeerCluster	Fügt einer Konfiguration mit mehreren Regionen einen Peer-Cluster hinzu	Ja
PutMultiRegionProperties	Legt Eigenschaften für mehrere Regionen für einen Cluster fest	Ja
PutWitnessRegion	Legt die Zeugenregion für einen Cluster mit mehreren Regionen fest	Ja

Ressourcenbasierte Richtlinie APIs

API-Operation	Description	Unterstützt RBP
DeleteClusterPolicy	Löscht die ressourcenbasierte Richtlinie aus einem Cluster	Ja
GetClusterPolicy	Ruft die ressourcenbasierte Richtlinie für einen Cluster ab	Ja
PutClusterPolicy	Erstellt oder aktualisiert die ressourcenbasierte Richtlinie für einen Cluster	Ja

AWS Fault Injection Service APIs

API-Operation	Description	Unterstützt RBP
InjectError	Fügt Fehler beim Testen von Fehlerinjektionen ein	Nein

Backup und Wiederherstellung APIs

API-Operation	Description	Unterstützt RBP
GetBackupJob	Ruft Informationen über einen Backup-Job ab	Nein
GetRestoreJob	Ruft Informationen über einen Wiederher stellungsauftrag ab	Nein
StartBackupJob	Startet einen Backup-Job für einen Cluster	Ja
StartRestoreJob	Startet einen Wiederherstellungsauftrag aus einem Backup	Nein
StopBackupJob	Stoppt einen laufenden Backup-Job	Nein
StopRestoreJob	Stoppt einen laufenden Wiederherstellungs auftrag	Nein

Verwenden von serviceverknüpften Rollen in Aurora DSQL

Aurora DSQL verwendet AWS Identity and Access Management (IAM) serviceverknüpfte Rollen. Eine serviceverknüpfte Rolle ist ein spezieller Typ einer 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 zur Verwendung von Aurora DSQL nicht manuell hinzufügen müssen. Wenn Sie einen Cluster erstellen, erstellt Aurora DSQL automatisch die serviceverknüpfte Rolle für Sie. Sie können die serviceverknüpfte Rolle erst löschen, nachdem Sie alle Cluster gelöscht haben. Dies schützt Ihre Aurora DSQL-Ressourcen, da Zugriffsberechtigungen nicht versehentlich für den Zugriff auf die Ressourcen entfernt werden 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.

Serviceverknüpfte Rollen sind in allen unterstützten Aurora DSQL-Regionen verfügbar.

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

Weitere Informationen finden Sie unter Serviceverknüpfte 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 haben Sie keine ungenutzte Entität, die nicht aktiv überwacht oder verwaltet wird.

Bevor Sie die serviceverknüpfte Rolle aus einem Konto löschen können, müssen Sie alle Cluster aus dem 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 unter Erstellen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

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 für die Verwendung von Bedingungsschlüsseln in den IAM-Berechtigungsrichtlinien.

Beispiel 1: Erteilen Sie die Erlaubnis, einen Cluster in einem bestimmten AWS-Region

Die folgende Richtlinie erteilt die Genehmigung zum Erstellen 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 AWS-Region s zu erstellen

Die folgende Richtlinie erteilt die Genehmigung zum Erstellen 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 einzugrenzen, sodass Aurora DSQL Cluster mit mehreren Regionen nur dann 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 Berechtigungen `AddPeerCluster`, `PutMultiRegionProperties` und `PutWitnessRegion` und in den 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 der Berechtigung zum Erstellen eines Clusters mit mehreren Regionen mit einer bestimmten Witness-Region

Die folgende Richtlinie verwendet einen Aurora DSQL-`dsql:WitnessRegion`-Bedingungsschlüssel und ermöglicht dem Benutzer, Cluster mit mehreren Regionen mit einer Witness-Region in USA West (Oregon) zu erstellen. Wenn Sie die `dsql:WitnessRegion`-Bedingung nicht angeben, können Sie eine beliebige Region als Witness-Region 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"
                    ]
                }
            }
        }
    ]
}
```

Vorfallreaktion in Amazon Aurora DSQL

Sicherheit hat bei AWS 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 Shared-Responsibility-Modells für die Reaktion auf Sicherheitsvorfälle auf Ihrer Seite verantwortlich.

Indem Sie eine Sicherheitsgrundlage schaffen, die den Anforderungen Ihrer in der Cloud betriebenen Anwendungen entspricht, können Sie Abweichungen erkennen und entsprechend darauf reagieren. Um besser zu verstehen, wie sich Ihre Entscheidungen und die Reaktion auf Sicherheitsvorfälle auf Ihre Unternehmensziele auswirken, empfehlen wir Ihnen, die folgenden Ressourcen zu prüfen:

- 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. Der Dienst überwacht Aktivitäten wie ungewöhnliche API-Aufrufe oder potenziell unbefugte Bereitstellungen, die auf eine mögliche Kompromittierung von Konten oder Ressourcen oder auf Auskundschaften durch Angreifer hinweisen. 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.

Compliance-Validierung 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. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Ausfallsicherheit in Amazon Aurora DSQL

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones (AZ). AWS-Regionen stellen mehrere physisch getrennte und isolierte Availability Zones bereit, 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 bieten Cluster mit einer

Region in Aurora DSQL 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 gleichzeitig äußerst konsistente Datenbankverfügbarkeit – sogar wenn Anwendungsclients nicht auf AWS-Region zugreifen können.

Weitere Informationen zu Availability Zones AWS-Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Zusätzlich zur AWS globalen Infrastruktur bietet Aurora DSQL mehrere Funktionen, um Ihre Datenstabilität und Backup-Anforderungen zu erfüllen.

Backup und Wiederherstellung

Aurora DSQL unterstützt Backup und Wiederherstellung mit AWS-Backup-Konsole. Sie können einen vollständigen Backup und eine 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 Witness-Region unterstützt ausschließlich Transaktionsprotokoll-Schreibvorgänge und verbraucht keinen Speicherplatz. Witness-Regionen haben keinen Endpunkt. Das bedeutet, dass Witness-Regionen nur verschlüsselte Transaktionsprotokolle speichern, keine Verwaltung oder Konfiguration erfordern, und für Benutzer nicht zugänglich sind. Wenn die Zeugenregion beeinträchtigt wird, hat dies keine Auswirkungen auf die Cluster-Verfügbarkeit. Bei Schreibtransaktionen kann es zu einem leichten Anstieg der Latenz kommen, bis sich die Zeugenregion erholt.

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, repliziert und führt automatisch Daten auf der Grundlage des Primärschlüsselbereichs der Datenbank und der Zugriffsmuster zusammen. Basierend auf der Lesezugriffshäufigkeit skaliert Aurora DSQL Lesereplikate automatisch nach oben und unten.

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, werden sie automatisch wieder dem Speicherquorum hinzugefügt und Ihrem Cluster zur Verfügung gestellt.

Hohe Verfügbarkeit

Standardmäßig sind Cluster mit einer Region und mehreren Regionen in Aurora DSQL aktiv-aktiv, und es müssen keine Cluster manuell bereitgestellt, konfiguriert oder neu konfiguriert werden. 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 in jede Regione lesen oder schreiben können, ohne dass eine 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.

Testen mit Fehlerinjektionen

Amazon Aurora DSQL ist in AWS Fault Injection Service (AWS FIS) integriert, einen vollständig verwalteten Service zur Durchführung kontrollierter Fehlerinjektionsexperimente, um die Widerstandsfähigkeit einer Anwendung zu verbessern. Mit Hilfe AWS FIS können Sie:

- Versuchsvorlagen erstellen, die spezifische Fehlerszenarien definieren
- Fehler injizieren (erhöhte Fehlerraten bei Clusterverbindungen), um die Mechanismen zur Behandlung und Wiederherstellung von Anwendungsfehlern zu validieren

- Testen Sie das Verhalten von Anwendungen in mehreren Regionen, um zu überprüfen, ob sich der Datenverkehr zwischen Anwendungen verschiebt, AWS-Regionen wenn hohe Verbindungsfehlerraten auftreten AWS-Region

In einem Cluster mit mehreren Regionen, der sich über die Regionen USA Ost (Nord-Virginia) und USA Ost (Ohio) erstreckt, können Sie beispielsweise ein Experiment in USA Ost (Ohio) durchführen, um dort Fehler zu testen, während USA Ost (Nord-Virginia) den normalen Betrieb fortsetzt. Diese kontrollierten Tests helfen Ihnen dabei, potenzielle Probleme zu identifizieren und zu lösen, bevor sie sich auf den Produktions-Workload auswirken.

Eine vollständige Liste der AWS FIS unterstützten Aktionen finden Sie im AWS FIS Benutzerhandbuch unter Aktionsziele.

Informationen zu Amazon Aurora DSQL-Aktionen, die in verfügbar sind AWS FIS, finden Sie in der Aurora DSQL-Aktionsreferenz im AWS FIS Benutzerhandbuch.

Informationen zu den ersten Schritten mit der Durchführung von Fehlerinjektionsexperimenten finden Sie unter Planung Ihrer AWS FIS -Experimente im AWS FIS -Benutzerhandbuch.

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 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 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 On-Premises-Netzwerk verbinden, finden Sie im 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 beim Erstellen des Aurora DSQL-Schnittstellenendpunkts nicht die private DNS-Option. Wenn private DNS aktiviert ist, werden Anfragen an den Aurora DSQL-Dienst, die aus Ihrer Amazon VPC gestellt werden, automatisch zur privaten IP-Adresse des Amazon VPC-Endpunkts aufgelöst – und nicht zum öffentlichen DNS-Namen. Wenn private DNS aktiviert ist, werden Aurora DSQL-Anfragen, die innerhalb Ihrer Amazon VPC gestellt werden, automatisch zum 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.

Auflisten von Clustern mithilfe einer Endpunkt-URL

Ersetzen Sie im folgenden Beispiel den AWS-Region `us-east-1` und den DNS-Namen der Amazon VPC-Endpunkt-ID `vpce-1a2b3c4d-5e6f.dsql.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

In der Aurora DSQL-API-Referenz finden Sie Informationen zur Ressourcenverwaltung in Aurora DSQL.

Verwalten von Endpunktrichtlinien

Durch gründliches Testen und Konfigurieren der Amazon VPC-Endpunktrichtlinien können Sie sicherstellen, dass Ihr Aurora DSQL-Cluster sicher, konform und auf die spezifischen Anforderungen Ihrer Organisation in Bezug auf Zugriffskontrolle und Governance abgestimmt ist.

Beispiel: Vollständige Aurora DSQL-Zugriffsrichtlinie

Die folgende Richtlinie gewährt Vollzugriff auf alle Aurora DSQL-Aktionen und -Ressourcen über den angegebenen Amazon VPC-Endpunkt.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

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: Rufen Sie den Dienstnamen für Ihren Cluster ab

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 Servicename enthält eine ID wie `dsql-fnh4` im Beispiel. Diese ID 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: Erstellen des Amazon VPC-Endpunkts

Erstellen Sie mit dem im vorherigen Schritt abgerufenen Servicenamen einen Amazon VPC-Endpunkt.

> **Important**
>
> Die folgenden Verbindungsanweisungen funktionieren nur für Clusterverbindungen, wenn Private DNS aktiviert ist. Verwenden Sie das `--no-private-dns-enabled`-Flag nicht, wenn Sie den Endpunkt erstellen, da sonst die unten aufgeführten Verbindungsanleitungen nicht ordnungsgemäß funktionieren. Wenn Sie Private 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);
```

Zusätzliche Einrichtung bei der Verbindung über Direct Connect oder Amazon VPC-Peering

Möglicherweise sind zusätzliche Einstellungen erforderlich, um über einen AWS PrivateLink Verbindungsendpunkt von lokalen Geräten über Amazon VPC-Peering oder eine Verbindung zu Aurora DSQL-Clustern herzustellen. Direct Connect Dieses Setup ist nicht erforderlich, wenn Ihre Anwendung in derselben Amazon VPC wie Ihr AWS PrivateLink Endpunkt läuft. Die oben erstellten privaten DNS-Einträge werden außerhalb der Amazon VPC des Endpunkts nicht korrekt aufgelöst. Sie können jedoch Ihre eigenen privaten DNS-Einträge erstellen, die zu Ihrem AWS PrivateLink Verbindungsendpunkt aufgelöst werden.

Erstellen Sie einen privaten CNAME-DNS-Eintrag, der auf den vollständig qualifizierten Domainnamen des AWS PrivateLink Endpunkts verweist. Der Domainname des erstellten DNS-Eintrags sollte aus den folgenden Komponenten bestehen:

1. Die Service-ID aus dem Servicenamen. Beispiel: `dsql-fnh4`
2. Der AWS-Region

Erstellen Sie den CNAME-DNS-Eintrag mit einem Domainnamen im folgenden Format: `*.service-identifier.region.on.aws`

Das Format des Domainnamens ist aus zwei Gründen wichtig:

1. Der Hostname, der für die Verbindung mit Aurora DSQL verwendet wird, muss mit dem Serverzertifikat von Aurora DSQL übereinstimmen, wenn der SSL-Modus verwendet wird`verify-full`. Dies gewährleistet ein Höchstmaß an Verbindungssicherheit.
2. Aurora DSQL verwendet den Cluster-ID-Teil des Hostnamens, der für die Verbindung mit Aurora DSQL verwendet wird, um den Verbindungscluster zu identifizieren.

Wenn das Erstellen von privaten DNS-Einträgen nicht möglich ist, können Sie trotzdem eine Verbindung zu Aurora DSQL herstellen. Siehe Herstellen einer Verbindung zu einem Aurora DSQL-Cluster über einen AWS PrivateLink Endpunkt ohne privates DNS.

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.

Konstruieren des Hostnamens

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 Service-ID aus dem Servicenamen. Beispiel: `dsql-fnh4`

3. Der. AWS-Region Beispiel: `us-east-1`

Verwenden Sie das folgende Format: *`cluster-id`*`.`*`service-identifier`*`.`*`region`*`.on.aws`

Beispiel: Verbindung über 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
```

Herstellen einer Verbindung zu einem Aurora DSQL-Cluster über einen AWS PrivateLink Endpunkt ohne privates DNS

Die obigen Verbindungsanweisungen basieren auf privaten DNS-Einträgen. Wenn Ihre Anwendung in derselben Amazon VPC wie Ihr AWS PrivateLink Endpunkt ausgeführt wird, werden die DNS-Einträge für Sie erstellt. Wenn Sie über Amazon VPC-Peering oder von lokalen Geräten aus eine Verbindung herstellen Direct Connect, können Sie alternativ Ihre eigenen privaten DNS-Einträge erstellen. Die Einrichtung von DNS-Einträgen ist jedoch aufgrund von Netzwerkbeschränkungen, die von Ihren Sicherheitsteams auferlegt wurden, nicht immer möglich. Wenn Ihre Anwendung eine Verbindung über Direct Connect oder von einer Peering-Amazon-VPC herstellen muss und die Einrichtung eines DNS-Eintrags nicht möglich ist, können Sie trotzdem eine Verbindung zu Aurora DSQL herstellen.

Aurora DSQL verwendet den Cluster-ID-Teil Ihres Hostnamens, um den verbindenden Cluster zu identifizieren. Wenn die Einrichtung eines DNS-Eintrags jedoch nicht möglich ist, unterstützt Aurora DSQL die Angabe des Zielclusters mithilfe der `amzn-cluster-id` Verbindungsoption. Mit dieser Option ist es möglich, den vollständig qualifizierten Domainnamen Ihres AWS PrivateLink Endpunkts als Hostnamen zu verwenden, wenn Sie eine Verbindung herstellen.

> **Important**
>
> Wenn Sie eine Verbindung mit dem vollständig qualifizierten Domainnamen oder der IP-Adresse Ihres AWS PrivateLink Endpunkts herstellen, wird der `verify-full` SSL-Modus nicht unterstützt. Aus diesem Grund wird die Einrichtung von privatem DNS bevorzugt.

Beispiel: Angabe der Cluster-ID-Verbindungsoption mit PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export HOSTNAME=vpce-04037adb76c111221-d849uc2p.dsql-fnh4.us-east-1.vpce.amazonaws.com
 # This should match your endpoint's fully-qualified domain name

# Construct the hostname used to generate the authentication token
export AUTH_HOSTNAME="$CLUSTERID.dsql.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $AUTH_HOSTNAME)

# Specify the amzn-cluster-id connection option
export PGOPTIONS="-c amzn-cluster-id=$CLUSTERID"

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Behebung von Problemen mit AWS PrivateLink

Häufige Probleme und Lösungen

In der folgenden Tabelle sind häufig auftretende Probleme und Lösungen im Zusammenhang mit AWS PrivateLink und Aurora DSQL aufgeführt.

Problem	Mögliche Ursache	Lösung
Verbindungstimeout	Die Sicherheitsgruppe ist nicht richtig konfiguriert	Verwenden Sie den Amazon VPC Reachability Analyzer, um sicherzustellen, dass Ihr

Problem	Mögliche Ursache	Lösung
		Netzwerk-Setup Datenverkehr auf Port 5432 zulässt.
Fehler für die DNS-Auflösung	Private DNS nicht aktiviert	Stellen Sie sicher, dass der Amazon VPC-Endpunkt mit aktiviertem Private DNS erstellt wurde.
Authentifizierungs fehler	Falsche Anmeldein formationen oder abgelaufenes Token	Generieren Sie ein neues Authentifizierungs token und überprüfen Sie den Benutzernamen.
Servicename nicht gefunden	Falsche Cluster-ID	Überprüfen Sie Ihre Cluster-ID und AWS-Regio n beim Abrufen des Dienstnamens.

Verwandte Ressourcen

Weitere Informationen finden Sie in den folgenden Ressourcen:

- Amazon Aurora DSQL-Benutzerhandbuch
- AWS PrivateLink -Dokumentation
- Greifen Sie auf Dienste zu über AWSAWS PrivateLink

Konfigurations- und Schwachstellenanalyse in Amazon Aurora DSQL

AWS erledigt grundlegende Sicherheitsaufgaben wie das Patchen von Gastbetriebssystemen (OS) und Datenbanken, die Firewallkonfiguration 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 kann AWS ein dienstübergreifender Identitätswechsel 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 Verwendung der globalen Bedingungskontext-Schlüssel `aws:SourceArn` und `aws:SourceAccount` in ressourcenbasierten Richtlinien, um die Berechtigungen, die Amazon Aurora DSQL einem anderen Service erteilt, auf eine bestimmte Ressource zu beschränken. 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 globalen Bedingungskontext-Schlüssel `aws:SourceArn` und `aws:SourceAccount` in Aurora DSQL verwenden können, um das Confused-Deputy-Problem zu vermeiden.

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 Methoden für die Aurora DSQL-Sicherheit

Aurora DSQL enthält eine Reihe von Sicherheitsfeatures, 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 detektivische Sicherheitsmethoden für Aurora DSQL
- Bewährte Methoden für vorbeugende Sicherheitsmaßnahmen für Aurora DSQL

Bewährte detektivische Sicherheitsmethoden 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.

Markieren 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 bestimmen.

- Vertraulichkeit – Eine Kennung für die spezifische Datenvertraulichkeitsebene, die eine Ressource unterstützt
- Umgebung – Wird verwendet, um zwischen Entwicklungs-, Test- und Produktionsinfrastruktur zu unterscheiden.

Weitere Informationen finden Sie unter Bewährte Methoden für das Markieren von AWS Ressourcen.

Bewährte Methoden für vorbeugende Sicherheitsmaßnahmen 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 von IAM-Rollen zum Authentifizieren des Zugriffs auf Aurora DSQL

Benutzer, Anwendungen und andere, AWS-Services die auf Aurora DSQL zugreifen, müssen gültige AWS Anmeldeinformationen in AWS API und AWS CLI Anfragen angeben. Sie sollten AWS Anmeldeinformationen nicht direkt in der Anwendung oder in den EC2-Instances speichern. Es handelt sich hierbei um langfristige Anmeldeinformationen, die nicht automatisch rotiert werden. Wenn diese Anmeldeinformationen kompromittiert werden, kann dies erhebliche geschäftliche Auswirkungen nach sich ziehen. 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 von IAM-Richtlinien für die Aurora DSQL-Basisautorisierung

Beim Erteilen von Berechtigungen entscheiden Sie, wer sie erhält, für welche Aurora DSQL-API-Operationen Berechtigungen erteilt werden 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.

Fügen Sie Berechtigungsrichtlinien zu IAM-Rollen hinzu und erteilen Sie Berechtigungen zur Ausführung von Operationen auf Aurora DSQL-Ressourcen. Weiterhin sind Berechtigungsgrenzen für IAM-Entitäten verfügbar, mit denen Sie die maximalen Berechtigungen festlegen, 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, benutzerdefinierte Datenbankrollen zu erstellen, um Ihren Cluster zu verwalten und eine Verbindung herzustellen. Weitere Informationen finden Sie unter Zugreifen auf Aurora DSQL und Grundlegendes zur Authentifizierung und Autorisierung für Aurora DSQL.

Einsatz von **`verify-full`** in Produktionsumgebungen

Diese Einstellung überprüft, ob das Serverzertifikat von einer vertrauenswürdigen Zertifizierungsstelle signiert wurde und ob der Server-Hostname mit dem Zertifikat übereinstimmt.

Aktualisieren Ihres PostgreSQL-Clients

Aktualisieren Sie Ihren PostgreSQL-Client regelmäßig auf die neueste Version, um von Sicherheitsverbesserungen zu profitieren. Wir empfehlen die Verwendung von PostgreSQL Version 17.

Tagging 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 zugehörige Wert ebenfalls optional.

Sie können die AWS-Managementkonsole, die AWS CLI oder die AWS-SDKs verwenden, um Tags für Aurora DSQL-Cluster hinzuzufügen, aufzulisten und zu löschen. Tags lassen sich sowohl während der Cluster-Erstellung als auch nachträglich über die AWS-Konsole hinzufügen. Um einen Cluster nach der Erstellung über die AWS CLI zu taggen, verwenden Sie die Operation `TagResource`.

Cluster mit einem Namen taggen

Aurora DSQL erstellt Cluster mit einer global eindeutigen ID, die als Amazon-Ressourcenname (ARN) zugewiesen wird. Wenn Sie Ihrem Cluster einen benutzerfreundlichen Namen geben möchten, empfiehlt es sich, dafür einen Tag zu verwenden.

Wenn Sie einen Cluster über die Aurora DSQL-Konsole erstellen, erzeugt Aurora DSQL automatisch einen Tag. Dieses Tag hat den Schlüssel Name und einen automatisch generierten Wert, der den Namen des Clusters repräsentiert. Dieser Wert ist konfigurierbar, sodass Sie Ihrem Cluster einen aussagekräftigeren Namen zuweisen können. Wenn ein Cluster einen Name-Tag und einen zugehörigen Wert hat, können Sie den Wert überall in der 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, Ziffern, Leerzeichen sowie die folgenden Sonderzeichen: _ . : / = + - @.
- Bei Schlüsseln und Werten wird die Groß-/Kleinschreibung berücksichtigt.

Hinweise zur Verwendung von Tags

Beim Einsatz von Tags in Aurora DSQL sollten Sie Folgendes beachten:

- Wenn Sie die AWS CLI oder Aurora DSQL-API-Operationen verwenden, geben Sie unbedingt den Amazon-Ressourcennamen (ARN) der betreffenden Aurora DSQL-Ressource an. Weitere Informationen finden Sie unter Amazon-Ressourcennamen(ARN)-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.
- Tags können beim Erstellen einer Ressource hinzugefügt werden, und Sie können Tags mithilfe der API-Operationen `TagResource`, `UntagResource` und `ListTagsForResource` anzeigen und bearbeiten.
- Tags können in IAM-Richtlinien verwendet werden. Mit Tags können Sie den Zugriff auf Aurora DSQL-Cluster verwalten zu steuern, welche Aktionen auf diese Ressourcen angewendet werden dürfen. Weitere Informationen finden Sie unter Zugriff auf AWS-Ressourcen mithilfe von Tags steuern.
- Sie können Tags auch für verschiedene andere Aktivitäten in AWS verwenden. Weitere Informationen finden Sie unter Häufig verwendete Tagging-Strategien.

Überlegungen zur Arbeit mit Amazon Aurora DSQL

Beachten Sie die folgenden Verhaltensweisen, wenn Sie mit Amazon Aurora DSQL arbeiten. Weitere Informationen über die Kompatibilität und Unterstützung von PostgreSQL finden Sie unter Kompatibilität der SQL-Features in Aurora DSQL. Informationen über Kontingente und Limits finden Sie unter Cluster-Kontingente und Datenbanklimits in Amazon Aurora DSQL.

- Die Berechnung des Speicherlimits kann einige Zeit in Anspruch nehmen, um den nach der Ausführung eines `DROP TABLE` Befehls freigegebenen Speicherplatz widerzuspiegeln. Wenn Sie zusätzliche Speicherkapazität benötigen, finden Sie weitere Informationen unter Cluster-Kontingente So fordern Sie Kontingentaktualisierungen an.
- Verwenden Sie für große Tabellen in Aurora DSQL den Systemkatalog, um die Anzahl der Tabellenzeilen anstelle von `COUNT(*)` Operationen abzurufen. Weitere Informationen finden Sie unter Verwenden von Systemtabellen und Befehlen in Aurora DSQL.
- Aurora DSQL verwaltet Berechtigungen über Zuschüsse auf Schemaebene. Admin-Benutzer erstellen Schemas mithilfe von `CREATE SCHEMA` und gewähren anderen Rollen Zugriff auf diese. `GRANT USAGE ON SCHEMA` Admin-Benutzer verwalten Objekte im öffentlichen Schema, während Benutzer ohne Administratorrechte Objekte in von Benutzern erstellten Schemas erstellen. Die Administratorrolle kann sich selbst jede andere Rolle zuweisen, um Berechtigungen für von Benutzern erstellte Objekte zu erhalten. Weitere Informationen finden Sie unter Autorisieren von Datenbankrollen zur Verwendung von SQL in Ihrer Datenbank.
- Wenn Treiber anrufen`PG_PREPARED_STATEMENTS`, bietet Aurora DSQL eine clusterweite Ansicht der zwischengespeicherten vorbereiteten Anweisungen. Möglicherweise sehen Sie pro Verbindung mehr vorbereitete Anweisungen als erwartet für denselben Cluster und dieselbe IAM-Rolle. Aurora DSQL verwaltet Anweisungsnamen während der Vorbereitung dynamisch.
- Stellen Sie beim Herstellen IPv4 einer Verbindung von Instanzen aus sicher, dass Ihr Client für IPv4 Verbindungen konfiguriert ist. Einige PostgreSQL-Clients versuchen, IPv4 sowohl IPv6 Verbindungen als auch Verbindungen im Dual-Stack-Modus herzustellen. Wenn es bei der IPv4 Verbindung zu einer Drosselung kommt, versucht der Client möglicherweise, auf Hosts, die nur diese Option verwenden, einen IPv6 Fehler zurückzugeben. `NetworkUnreachable` IPv4 Konfigurieren Sie Ihren Client so, dass er IPv4 explizit verwendet wird, um dieses Verhalten zu vermeiden.
- Nachdem ein Admin-Benutzer ein neues Schema erstellt hat `GRANT` und die `REVOKE` Änderungen innerhalb der Verbindungsdauer (bis zu einer Stunde) auf bestehende Verbindungen übertragen werden. Stellen Sie nach Änderungen der Berechtigungen sofort eine neue Verbindung her.

- In seltenen Wiederherstellungsszenarien für verknüpfte Cluster mit mehreren Regionen sorgen automatisierte Clusterwiederherstellungsvorgänge für eine hohe Verfügbarkeit, es kann jedoch zu vorübergehenden Parallelitätskontrollen oder Verbindungsfehlern kommen. In den meisten Fällen ist nur ein Prozentsatz Ihrer Arbeitslast betroffen. Wenn Sie auf diese vorübergehenden Fehler stoßen, wiederholen Sie Ihre Transaktion oder stellen Sie erneut eine Verbindung zu Ihrem Kunden her.
- Einige SQL-Clients, wie Datagrip, fordern umfangreiche Systemmetadaten an, um Schemainformationen aufzufüllen. Aurora DSQL bietet Kernmetadaten für SQL-Abfragefunktionen. Bei der Schemaanzeige auf diesen Clients werden im Vergleich zum vollen Funktionsumfang möglicherweise nur begrenzte Informationen angezeigt.
- Um sicherzustellen, dass Abfragen neu erstellte Schemas und Tabellen erkennen, aktualisieren Sie Ihre Verbindung, nachdem Sie Datenbankobjekte erstellt oder gelöscht haben. Dies schließt Szenarien ein, in denen nach dem Löschen eines Schemas oder beim Abfragen von Objekten, die in einer anderen Verbindung erstellt wurden, `Schema Already Exists` Fehler auftreten. Trennen Sie die Verbindung und stellen Sie die Verbindung wieder her, oder führen Sie den Vorgang `SET search_path` erneut aus, um den Katalog-Cache zu aktualisieren.
- Verwenden Sie dies bei komplexen Abfragen, `EXPLAIN ANALYZE VERBOSE` um Vorgänge mit hoher Latenz zu identifizieren und Abfragepläne zu optimieren. Durch das Abdecken von Indizes können die DPU-Kosten erheblich gesenkt werden, indem nur Indexscans anstelle von vollständigen Tabellenscans aktiviert werden. Weitere Informationen finden Sie unter Mit Aurora DSQL EXPLAIN-Plänen arbeiten.
- Verbindungslimits werden auf Clusterebene verwaltet. Informationen Cluster-Kontingente zur Anforderung von Kontingentaktualisierungen finden Sie unter.

Cluster-Kontingente und Datenbanklimits 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

Description	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`
Maximales Speicherk ontingent pro Cluster	10 TiB Standardl imit, bis zu 256 TiB mit genehmigt er Grenzwert erhöhung	Ja	`DISK_FULL(53100)`

Description	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode
Maximale Verbindungsanzahl 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.

Description	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 Anzahl an Zeilen 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

Description	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 werden	10 MiB	Nein	`54000`	`ERROR: transaction size limit DETAIL: Current transaction s 10mb`
Maximale Anzahl von Tabellenzeilen, die in einem Transaktionsblock mutiert werden können	3 000 Zeilen pro Transaktion Siehe Überlegungen zur PostgreSQL-Kompatibilität mit Aurora DSQL.	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.`

Description	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode	Fehlermeldung
Maximale Anzahl von Schemas, die in einer Datenbank definiert werden können	10	Nein	54000	ERROR: more than 10 schemas n
Maximale Anzahl an Tabellen in einer Datenbank	1 000 Tabellen	Nein	54000	ERROR: creating more than 100 allowed
Maximale Anzahl von Datenbanken pro Cluster	1	Nein	Kein Fehlercode	ERROR: unsupported statement
Maximale Anzahl von 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

Description	Standardlimit	Konfigurierbar?	Aurora DSQL-Fehlercode	Fehlermeldung
Maximale Anzahl von Sequenzen	5,000	Nein	`54000`	`ERROR: creating more than 500 not allowed`

Informationen zu Datentypbeschränkungen für Aurora DSQL finden Sie unter Unterstützte Datentypen in Aurora DSQL.

Aurora DSQLAPI-Referenz

Neben dem AWS-Managementkonsole und dem AWS Command Line Interface (AWS CLI) bietet Aurora DSQL auch eine API-Schnittstelle. Sie können die API-Operationen zur Verwaltung Ihrer Ressourcen in Aurora DSQL verwenden.

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 zur AWS CLI finden Sie in der AWS Command Line Interface-Referenz für Aurora DSQL.

Fehlerbehebung bei Problemen mit Aurora DSQL

> **Note**
>
> Die folgenden Themen bieten Hinweise zur Fehlerbehebung bei Problemen und Fehlermeldungen, die bei der Verwendung von Aurora DSQL auftreten können. Falls Sie auf ein Problem stoßen, das hier nicht aufgeführt ist, wenden Sie sich bitte an den AWS -Support.

Themen

- Fehlerbehebung bei Verbindungsfehlern
- Problembehebung bei Authentifizierungsfehlern
- Fehlerbehebung bei Autorisierungsfehlern
- Fehlerbehebung bei SQL-Fehlern
- Fehlerbehebung bei OCC-Fehlern
- Problembehandlung bei Verbindungen SSL/TLS

Fehlerbehebung bei Verbindungsfehlern

Fehler: unbekannter SSL-Fehlercode: 6 oder Verbindung konnte nicht akzeptiert werden, SNI wurde nicht empfangen

Möglicherweise verwenden Sie eine PSQL-Version vor Version 14, die Server Name Indication (SNI) nicht unterstützt. SNI ist für die Verbindung mit Aurora DSQL erforderlich.

Sie können Ihre Client-Version mit dem Befehl `psql --version` überprüfen.

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 häufig bei IPv4 Nur-Instances auf, da PostgreSQL-Clients Dual-Stack-Verbindungen verarbeiten. 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.

Problembehebung bei Authentifizierungsfehlern

Die IAM-Authentifizierung ist für den Benutzer „...“ fehlgeschlagen

Wenn Sie ein IAM-Authentifizierungstoken in Aurora DSQL generieren, können Sie eine maximale Gültigkeit von 1 Woche festlegen. Nach einer Woche können Sie sich nicht mehr mit diesem Token authentifizieren.

Darüber hinaus lehnt Aurora DSQL Ihre Verbindungsanfrage ab, wenn Ihre angenommene Rolle abgelaufen ist. Wenn Sie sich beispielsweise mit einer temporären IAM-Rolle verbinden möchten lehnt Aurora DSQL die Verbindungsanfrage ab, auch wenn Ihr Authentifizierungstoken noch gültig ist.

Weitere Informationen zur Funktionsweise von IAM in Verbindung mit Aurora DSQL finden Sie unter Grundlegendes zur Authentifizierung und Autorisierung für Aurora DSQL und unter 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 werden Anforderungen signiert?.

Die IAM-Rolle <role> ist nicht vorhanden

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 — IAM ARNs.

Falsche Zuordnung von Benutzer zu Aktion

Dieser Fehler tritt auf, wenn der Authentifizierungstokentyp nicht mit der Datenbankrolle übereinstimmt. Aurora DSQL verwendet zwei Tokentypen: `DbConnectAdmin` für die `admin` Rolle und `DbConnect` für benutzerdefinierte Datenbankrollen.

- Wenn Sie sehen`Wrong user to action mapping. user: admin, action: DbConnect`, verwenden Sie `generate-db-connect-admin-auth-token` statt. `generate-db-connect-auth-token`

- Wenn Sie sehen`Wrong user to action mapping. user: myusername, action: DbConnectAdmin`, verwenden Sie `generate-db-connect-auth-token` stattgenerate-db-connect-admin-auth-token.

Fehlerbehebung bei Autorisierungsfehlern

Rolle <role> wird nicht unterstützt

Aurora DSQL unterstützt die Operation `GRANT` nicht. Siehe Unterstützte Teilmengen von SQL-Befehlen in Aurora DSQL.

Vertrauensstelle mit Rolle <role> kann nicht hergestellt werden

Aurora DSQL unterstützt die Operation `GRANT` nicht. Siehe Unterstützte Teilmengen von SQL-Befehlen in Aurora DSQL.

Die Rolle <role> ist nicht vorhanden

Aurora DSQL konnte den angegebenen Datenbankbenutzer nicht finden. Weitere Informationen finden Sie unter Autorisieren benutzerdefinierter Datenbankrollen für die Verbindung zu einem Cluster.

FEHLER: Berechtigung für die Gewährung einer IAM-Vertrauensstellung für <role> wurde verweigert

Um einer Datenbankrolle Zugriff zu gewähren, müssen Sie in der Admin-Rolle mit Ihrem Cluster verbunden sein. Weitere Informationen finden Sie unter Datenbankrollen für die SQL-Nutzung autorisieren.

FEHLER: die Rolle <role> muss das LOGIN-Attribut haben

Jede Datenbankrolle, die Sie erstellen, muss über die `LOGIN`-Berechtigung verfügen.

Um diesen Fehler zu beheben, stellen Sie sicher, dass Sie die PostgreSQL-Rolle mit der Berechtigung `LOGIN` erstellt haben. Weitere Informationen finden Sie unter ROLLE ERSTELLEN und GEWÄHREN in der PostgreSQL-Dokumentation.

FEHLER: die Rolle <role> kann nicht gelöscht werden, da einige Objekte davon abhängen

Aurora DSQL gibt einen Fehler zurück, wenn Sie versuchen, eine Datenbankrolle mit einer IAM-Beziehung zu löschen, bevor Sie diese Beziehung mit `AWS IAM REVOKE` widerrufen haben. Weitere Informationen finden Sie unter Autorisierung widerrufen.

Fehlerbehebung bei SQL-Fehlern

Fehler: nicht unterstützt

Aurora DSQL unterstützt nicht alle PostgreSQL-basierten Dialekte. Weitere Informationen finden Sie unter Unterstützte PostgreSQL-Features in Aurora DSQL.

Fehler: verwenden Sie stattdessen **`CREATE INDEX ASYNC`**

Um einen Index für eine Tabelle mit vorhandenen Zeilen zu erstellen, müssen Sie den Befehl `CREATE INDEX ASYNC` verwenden. Weitere Informationen finden Sie unter Asynchrone Erstellung von Indizes in Aurora DSQL.

Fehlerbehebung bei OCC-Fehlern

OC000 „FEHLER: Die Mutation steht in Konflikt mit einer anderen Transaktion, versuchen Sie es bei Bedarf erneut“

Diese Transaktion hat versucht, dieselben Tupel wie eine andere, gleichzeitige Transaktion zu ändern. Dies deutet auf einen Konflikt bei den modifizierten Tupeln hin. Weitere Informationen finden Sie unter Parallelitätssteuerung in Aurora DSQL

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. Die nennen wir 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 zu lesen versuchen, aktualisiert Aurora DSQL den Katalog-Cache. Die Transaktion bei T3 verwendet die Katalogversion V2. Aurora DSQL wird die Transaktion abschließen, sofern seit dem Zeitpunkt T2 keine weiteren Katalogänderungen vorgenommen wurden.

Problembehandlung bei Verbindungen SSL/TLS

SSL-Fehler: Zertifikatverifizierung fehlgeschlagen

Dieser Fehler weist darauf hin, dass der Client das Serverzertifikat nicht verifizieren kann. Stellen Sie Folgendes sicher:

1. Das Amazon Root CA 1-Zertifikat ist ordnungsgemäß installiert. Anleitungen zur Validierung und Installation dieses Zertifikats finden Sie unter Konfiguration von SSL/TLS Zertifikaten für Aurora DSQL-Verbindungen.
2. Die `PGSSLROOTCERT`-Umgebungsvariable verweist auf die richtige Zertifikatsdatei.
3. Die Zertifikatsdatei verfügt über die richtigen Berechtigungen.

Unbekannter SSL-Fehlercode: 6

Dieser Fehler tritt bei PostgreSQL-Clients auf, die älter als Version 14 sind. 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. erwenden Sie die in der Eine Verbindung von Windows aus herstellen-Anleitung beschriebene Methode mit der heruntergeladenen Zertifikatdatei.

Feedback zu Amazon Aurora DSQL geben

Wenn Sie auf Funktionen stoßen, die für Ihre Migration wichtig sind, aber derzeit nicht in Aurora DSQL unterstützt werden, bietet AWS mehrere Kanäle für Feedback:

Feedback-Kanäle

Aurora DSQL Discord-Server

Treten Sie dem Aurora DSQL Discord-Server bei, um sich mit dem AWS-Team und der AWS-Community zu verbinden. Teilen Sie Funktionsanfragen, besprechen Sie Herausforderungen bei der Migration und erhalten Sie Feedback in Echtzeit.

AWS Support

Wenn Sie einen AWS-Supportplan haben, erstellen Sie einen Support-Fall, um Ihre spezifischen Anforderungen und Zeitpläne zu besprechen.

AWS re:POST

Verwenden Sie AWS re:POST, um Fragen zu stellen und Feedback mit der Community und AWS-Experten zu teilen.

Effektive Funktionsanfragen

Wenn Sie Funktionen anfordern, geben Sie Folgendes an:

- Beschreibung des Anwendungsfalls: Erläutern Sie, was Sie erreichen möchten und warum
- Aktuelle Problemumgehung: Beschreiben Sie alle Alternativen, die Sie ausprobiert haben
- Auswirkungen auf das Geschäft: Erläutern Sie, wie sich die fehlende Funktion auf Ihren Migrationszeitplan oder die Anwendungsfunktionalität auswirkt
- Prioritätsstufe: Geben Sie an, ob dies Ihre Migration blockiert oder eine nice-to-have Verbesserung darstellt

Dokumentverlauf für das Amazon Aurora DSQL-Benutzerhandbuch

In der folgenden Tabelle werden die Dokumentationsversionen für Aurora DSQL aufgelistet.

Änderung	Beschreibung	Datum
Neuer Inhalt: Aurora DSQL Connector für .NET Npgsql	Dokumentation für den Aurora DSQL Connector für .NET Npgsql hinzugefügt, der Npgsql mit automatischer IAM-Authentifizierung umhüllt. Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und die Verbindungsverwaltung für .NET-Anwendungen. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem.NET Npgsql-Konnektor.	20. März 2026
Aktualisierter Inhalt: SQL-Befehlsreferenz und Systemabfragen	START TRANSACTION und ROLLBACK wurden der Referenz für Transakti onssteuerungsbefehle hinzugefügt, wobei END und ABORT als Aliase verwendet wurden. Es wurden nützliche Systemabfragen zum Abrufen von Aurora DSQL- und PostgreSQL-Versionsinformat ionen hinzugefügt. Weitere Informationen finden Sie in der	13. März 2026

	PostgreSQL-Kompatibilitätsreferenz.	
Aktualisierter Inhalt: Daten in Aurora DSQL laden	Der Leitfaden zum Laden von Daten wurde mit Informationen zur clientseitigen `\copy` Verwendung, bewährten Methoden für INSERT und Anleitungen zur Vorab-Erstellung von Tabellen vor dem Laden aktualisiert. Weitere Informationen finden Sie unter Daten in Aurora DSQL laden.	13. März 2026
Neuer Inhalt: Aurora DSQL Connector für Ruby pg	Dokumentation für den Aurora DSQL Connector für Ruby pg hinzugefügt, der das PG-Gem mit automatischer IAM-Authentifizierung umhüllt. Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und das Verbindungsmanagement für Ruby-Anwendungen. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem Ruby-PG-Konnektor.	12. März 2026
Aktualisierter Inhalt: Asynchrone DDL-Jobs	Die `sys.jobs` Dokumentation wurde mit erweiterten Details zur Überwachung und Verwaltung asynchroner DDL-Operationen aktualisiert. Weitere Informationen finden Sie in der PostgreSQL-Kompatibilitätsreferenz.	6. März 2026

Aktualisierter Inhalt: Generierung von PHP-Authentifizierungstoken	Der Seite zur Generierung von Authentifizierungstoken wurde ein PHP-SDK-Tab hinzugefügt. Weitere Informationen finden Sie unter Generieren von Authentifizierungstoken.	5. März 2026
Neuer Inhalt: Daten in Aurora DSQL laden	Es wurde eine Anleitung zum Laden von Daten in Aurora DSQL-Cluster hinzugefügt, einschließlich der Verwendung des Aurora DSQL Loader-Dienstprogramms. Weitere Informationen finden Sie unter Daten in Aurora DSQL laden.	5. März 2026
Neuer Inhalt: Sequenzen und Identitätsspalten	Unterstützung für Sequenzen und Identitätsspalten hinzugefügt. Neue Referenzseiten für SQL-Befehle für CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE und Funktionen zur Sequenzmanipulation. CREATE TABLE und ALTER TABLE wurden aktualisiert und enthalten nun die Syntax für Identitätsspalten. Es wurde eine neue Anleitung zur Auswahl von Identifier-Typen und Cache-Größen hinzugefügt. Weitere Informationen finden Sie unter Sequenzen und Identitätsspalten.	11. Februar 2026

Neuer Inhalt: Aurora DSQL Connector für Go	Dokumentation für den Aurora DSQL Connector for Go hinzugefügt, der PGX mit automatischer IAM-Authentifizierung umhüllt. Der Connector kümmert sich um die Token-Generierung, die SSL-Konfiguration und das Verbindungsmanagement für Go-Anwendungen. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem Go-Connector.	5. Februar 2026
Aktualisierter Inhalt: Amazon Aurora DSQL-Cluster-Konnektivitätstools	Die Dokumentation zu den Cluster-Konnektivitätstools wurde neu organisiert, um den Unterschied zwischen den von ihnen AWS bereitgestellten Konnektoren, Adaptern und Tools von Drittanbietern zu verdeutlichen. Fehlende Links zu Codebeispielen hinzugefügt. Weitere Informationen finden Sie unter Amazon Aurora DSQL-Cluster-Konnektivitätstools.	26. Januar 2026

Neuer Inhalt: Aurora DSQL Plugin für DBeaver Community Edition	Dokumentation für das Aurora DSQL Plugin for DBeaver Community Edition hinzugefügt, das die IAM-Authentifizierung und die vereinfachte Verbindungseinrichtung für Aurora DSQL-Cluster ermöglicht. Enthält Installationsanweisungen, Verbindungskonfiguration und Anleitungen zur Fehlerbehebung. Weitere Informationen finden Sie unter Für DBeaver den Zugriff auf Aurora DSQL verwenden.	26. Januar 2026
Neuer Inhalt: Aurora DSQL-Treiber für SQLTools	Dokumentation für den Aurora DSQL-Treiber für hinzugefügt SQLTools, eine Visual Studio Code-Erweiterung, die es Entwicklern ermöglicht, mit automatischer IAM-Authentifizierung direkt von VS Code aus eine Verbindung zu Aurora-DSQL-Datenbanken herzustellen und diese abzufragen. Weitere Informationen finden Sie unter Verwenden des Aurora-DSQL-Treibers für SQLTools.	26. Januar 2026

Aktualisierter Inhalt: Aurora DSQL Steering: Skills and Powers	Dokumentation zur Unterstützung der Installation mit der Skills CLI für agentenunabhängigen Support hinzugefügt. Die Skills CLI bietet eine vereinfachte Einrichtungsmethode, die für mehrere KI-Codierungsassistenten wie Claude Code, Cursor, Copilot, Gemini und andere funktioniert. Weitere Informationen finden Sie unter Aurora DSQL Steering: Skills and Powers.	23. Januar 2026
Neuer Inhalt: Aurora DSQL Adapter für Tortoise ORM	Unterstützung für Tortoise ORM, ein asynchrones Python-ORM-Framework, wurde hinzugefügt. Der Aurora DSQL Adapter für Tortoise ORM ermöglicht es Entwicklern, Tortoise ORM mit Aurora DSQL-Clustern zu verwenden. Weitere Informationen finden Sie unter Aurora DSQL-Adapter und Dialekte.	23. Januar 2026

Neuer Inhalt: Aurora DSQL Steering: Fähigkeiten und Fähigkeiten	Neue Dokumentation für die Konfiguration der KI-Steuerung mit Aurora DSQL unter Verwendung von Fähigkeiten und Befugnissen hinzugefügt. Enthält Anweisungen zur Einrichtung von Kiro Powers, Claude Skills, Gemini Skills und Codex Skills. Weitere Informationen finden Sie unter Aurora DSQL Steering: Skills and Powers.	16. Januar 2026
Unterstützung für numerischen Datentypindex	Indexunterstützung für den `numeric` Datentyp in Aurora DSQL hinzugefügt. Sie können jetzt `numeric` Spalten als Primärschlüssel und in Sekundärindizes verwenden. Weitere Informationen finden Sie unter Unterstützte Datentypen in Aurora DSQL.	13. Januar 2026
Aktualisierter Inhalt: Unterstützte Teilmengen von SQL-Befehlen	Die Dokumentation zu den SQL-Befehlen wurde zur besseren Navigation und besseren Übersichtlichkeit in separate Seiten umstrukturiert. Jeder Befehl (CREATE TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW) hat jetzt eine eigene Seite. Weitere Informationen finden Sie unter Unterstützte Teilmengen von SQL-Befehlen.	6. Januar 2026

Aktualisierter Inhalt: AWS Labs Aurora DSQL MCP Server	Die MCP Server-Dokumentation wurde mit detaillierten Installationsansätzen für Claude Code und Codex aktualisiert, einschließlich Beispielen für CLI-basierte Setup- und Konfigurationsdateien. Es wurde eine umfassende Anleitung zum Auffinden von MCP-Client-Konfigurationsdateien in verschiedenen Entwicklungstools hinzugefügt. Weitere Informationen finden Sie unter AWS Labs Aurora DSQL MCP Server.	19. Dezember 2025
Aktualisierter Inhalt: Migration von PostgreSQL zu Aurora DSQL	Der Abschnitt zur PostgreSQL-Kompatibilität wurde als umfassender Migrationsleitfaden neu verfasst. Enthält Informationen zur Framework-Kompatibilität, allgemeine Migrationsmuster, architektonische Unterschiede und KI-gestützte Migrationsanleitungen. Es wurde ein neues Kapitel hinzugefügt, um Feedback zu Aurora DSQL zu geben. Weitere Informationen finden Sie unter Migration von PostgreSQL zu Aurora DSQL.	16. Dezember 2025

Aktualisierter Inhalt: Verbindung zu Aurora DSQL herstellen mit AWS PrivateLink	Es wurde eine Dokumentation für die private DNS-Einrichtung und die Cluster-ID-Verbindungsoption hinzugefügt, um Kunden zu unterstützen, die Amazon AWS PrivateLink VPC-Peering Direct Connect oder Amazon VPC-Peering verwenden. Enthält Anleitungen zum Herstellen einer Verbindung ohne privates DNS mithilfe der `amzn-cluster-id` Verbindungsoption. Weitere Informationen finden Sie unter Verwaltung und Verbindung zu Aurora DSQL-Clustern mithilfe von AWS PrivateLink.	11. Dezember 2025
Aktualisierter Inhalt: Aurora DSQL-Cluster-Lebenszyklus	Die Dokumentation für das Aurora DSQL-Cluster-Lebenszyklusmanagement wurde aktualisiert. Erläutert Clusterstatusdefinitionen, Zustandsübergänge und Operationen, die im Status „Inaktiv“ und „Inaktiv“ verfügbar sind. Weitere Informationen finden Sie unter Aurora DSQL-Cluster-Lebenszyklus.	4. Dezember 2025

Neuer Inhalt: Aurora DSQL-Konnektoren für Python und Node.js	Dokumentation für Aurora DSQL-Konnektoren für Python (psycopg, psycopg2, asyncpg) und Node.js (node-postgres, Postgres.js) hinzugefügt. Diese Konnektoren integrieren die IAM-Authentifizierung für die Verbindung von Anwendungen mit Aurora DSQL-Clustern. Weitere Informationen finden Sie unter Aurora DSQL-Konnektoren.	21. November 2025
Neuer Inhalt: Verwendung JupyterLab mit Aurora DSQL	step-by-stepAnleitung zum Verbinden und Abfragen von Aurora DSQL JupyterLab mithilfe von Python hinzugefügt. Enthält Anweisungen sowohl für lokale JupyterLab Installationen als auch für Amazon SageMaker AI-Umgebungen. Weitere Informationen finden Sie unter Verwenden JupyterLab mit Aurora DSQL.	20. November 2025
Aktualisierter Inhalt: Kontingente für Aurora DSQL	Das maximale Cluster-Speicherkontingent wurde von 128 TiB auf 256 TiB aktualisiert. Weitere Informationen finden Sie unter Kontingente für Aurora DSQL.	19. November 2025

Neuer Inhalt: Erste Schritte mit dem Aurora DSQL Query Editor	Dokumentation für die Verwendung des Aurora DSQL Query Editors in der AWS Management Console hinzugefügt. Enthält Voraussetzungen, Verbindungseinrichtung und Anweisungen zur Ausführung von Abfragen. Weitere Informationen finden Sie unter Erste Schritte mit dem Aurora DSQL Query Editor.	18. November 2025

Ressourcenbasierte Richtlinienunterstützung für Amazon Aurora DSQL	Unterstützung für ressourcenbasierte Richtlinien (RBP) mit neuen Berechtigungen hinzugefügt:, und. `PutClusterPolicy` `GetClusterPolicy` `DeleteClusterPolicy` Diese Berechtigungen ermöglichen die Verwaltung von Inline-Richtlinien, die an Aurora DSQL-Cluster angehängt sind, um eine differenzierte Zugriffskontrolle zu gewährleisten. Die verwalteten Richtlinien AmazonAurora DSQLFull wurden aktualisiert und bieten nun AmazonAurora DSQLRead OnlyAccess auch AmazonAurora DSQLConsole FullAccess RBP-Funktionen. Weitere Informationen finden Sie unter AWS Verwaltete Richtlinien für Amazon Aurora DSQL.	15. Oktober 2025

Aurora DSQL JDBC-Anschluss	Dokumentation für den Aurora DSQL JDBC Connector hinzugefügt, einen PGJDBC-Connector, der die IAM-Authentifizierung für die Verbindung von Java-Anwendungen mit Amazon Aurora DSQL-Clustern integriert. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora DSQL-Clustern mit einem JDBC-Konnektor.	2. September 2025
AWS verwaltete Richtlinienaktualisierungen für die Integration AWS FIS	Aktualisierte `AmazonAuroraDSQLFullAccess` `AmazonAuroraDSQLConsoleFullAccess` Richtlinien zur Unterstützung der AWS Fault Injection Service Integration mit Aurora DSQL. Damit können Sie Fehler in Aurora DSQL-Clustern mit einer oder mehreren Regionen gezielt auslösen, um die Fehlertoleranz Ihrer Anwendungen zu testen. Weitere Informationen zu diesen Richtlinien finden Sie unter Von AWS verwaltete Richtlinienaktualisierungen.	19. August 2025

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	Es wurde die Möglichkeit hinzugefügt, Backup- und Wiederherstellungsoperationen für Aurora DSQL-Cluster durchzuführen – einschließlich Starten, Stoppen und Überwachen von Aufträgen. Weiterhin wurde die Möglichkeit hinzugefügt, kundenseitig verwaltete KMS-Schlüssel für die Clusterverschlüsselung zu verwenden. Weitere Informationen finden Sie unter AmazonAuroraDSQLFullZugreifen und Verwenden von serviceverknüpften Rollen in Aurora DSQL.	21. Mai 2025

AmazonAuroraDSQLConsoleFullAccess update	Ermöglicht jetzt die Durchführung von Sicherungs- und Wiederherstellungsvorgänge für Aurora DSQL-Cluster über die AWS Console Home. Dies beinhaltet das Starten, Stoppen und Überwachen von Aufträgen. Weiterhin wird jetzt auch die Verwendung von kundenseitig verwalteten KMS-Schlüsseln für die Clusterverschlüsselung und den AWS CloudShell-Start unterstützt. Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von servicevorknüpften Rollen in Aurora DSQL.	21. Mai 2025

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 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 zur Steuerung, 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	Neue Berechtigungen wurden Aurora DSQL hinzugefügt, um Clustermanagement in mehreren Regionen und VPC-Endpunktverbindungen zu unterstützen. Zu den neuen Berechtigungen gehören: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` , `RemovePeerCluster` und `GetVpcEndpointServiceName` . Weitere Informationen finden Sie unter AmazonAuroraDSQLConsoleFullAccessund Verwenden von serviceverknüpften Rollen in Aurora DSQL.	13. Mai 2025

AuroraDsqlServiceLinkedRolePolicy update	Es wurde die Möglichkeit hinzugefügt, Metriken in den `AWS/AuroraDSQL` - und `AWS/Usage CloudWatch` - Namespaces der Richtlinie zu veröffentlichen. Auf diese Weise kann der zugehörige Dienst oder die zugehörige Rolle umfassendere Nutzungs- und Leistungsdaten an Ihre CloudWatch Umgebung weitergeben. 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 Verwalten und Verbinden von Amazon Aurora DSQL-Clustern mithilfe von AWS PrivateLink.	8. Mai 2025

Erstversion	Erstveröffentlichung des Amazon Aurora DSQL-Benutzerhandbuchs.	03. 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.