aws

Guide de l’utilisateur

Amazon Aurora DSQL

Amazon Aurora DSQL: Guide de l’utilisateur

Table of Contents

Qu'est-ce qu'Amazon Aurora DSQL ? 1
Quand l'utiliser 1
Fonctions principales 1
Région AWS disponibilité 3
Clusters multi-régions 5
Tarification 6
Quelle est la prochaine étape ? 6
Mise en route 7
Conditions préalables 7
Création d'un cluster à une seule région 7
Connexion à un cluster 8
Exécution de commandes SQL 9
Création d'un cluster multi-régions 10
Résolution des problèmes 12
Authentification et autorisation 13
Gestion de votre cluster 13
Connexion à votre cluster 13
Rôles PostgreSQL et IAM 14
Utilisation des actions de politique IAM avec Aurora DSQL 15
Utilisation des actions de politique IAM pour se connecter aux clusters 16
Utilisation des actions de politique IAM pour gérer les clusters 16
Révocation d'une autorisation à l'aide d'IAM et de PostgreSQL 17
Création d'un jeton d'authentification 18
Console 19
AWS CloudShell 19
AWS CLI 21
Aurora SQL SDKs 22
Rôles de base de données et authentification IAM 31
Rôles IAM 31
Utilisateurs IAM 32
Connexion 32
Query 32
Affichage des mappages 33
Révocation 34

Aurora DSQL et PostgreSQL 35
Points forts de la compatibilité 35
Avantages de l'architecture distribuée 36
Compatibilité avec SQL 37
Types de données pris en charge 37
Fonctionnalités SQL prises en charge 43
Sous-ensembles de commandes SQL pris en charge 47
Guide de migration 68
Contrôle de la simultanéité 76
Conflits de transaction 77
Directives pour optimiser les performances des transactions 77
Transactions DDL et distribuées 78
Clés primaires 79
Structure et stockage des données 79
Directives pour choisir une clé primaire 80
Séquences et colonnes d'identité 81
Fonctions de manipulation de séquence 81
Colonnes d'identité 84
Utilisation de séquences et de colonnes d'identité 86
Index asynchrones 87
Syntaxe 88
Parameters 88
Notes d'utilisation 89
Création d'un index 90
Interrogation d'un index 91
Défaillances de création d'index uniques 92
Violations d'unicité 92
Tables et commandes système 95
Tables système 95
Requêtes système utiles 105
Commande ANALYZE. 106
EXPLIQUEZ LES PLANS 107
Plans PostgreSQL EXPLAIN 108
Eléments clés 109
Le filtrage 109
Lire les plans EXPLAIN 110

DPUs dans EXPLAIN ANALYZE 114
Gestion des clusters Aurora DSQL 119
Clusters à une seule région 119
Utilisation des kits SDK AWS 119
Utilisation de l'interface de ligne de commande (CLI) AWS 158
Clusters multi-régions 161
Utilisation des kits SDK AWS 162
Utilisation de la AWS CLI 216
CloudFormation 222
Configuration initiale 222
Recherche de clusters 223
Mise à jour de la configuration 223
Cycle de vie du cluster SQL Aurora 224
États de cluster 224
Affichage de l'état des clusters 227
Programmation avec Aurora DSQL 228
Connecteurs 229
Connecteur JDBC 229
Connecteur Python 234
Connecteur Go 246
Connecteurs Node.js 253
Connecteur Ruby 262
Connecteur .NET 268
Accès à Aurora DSQL 274
Clients SQL 275
DBeaver 276
JetBrains DataGrip 279
Psql 281
VSCode 282
Résolution des problèmes 284
Outils de connectivité des bases de données 284
Adaptateurs Aurora DSQL 284
Exemples de pilotes de base de données 285
Exemples d'ORM et de framework 287
Chargement des données 288
Choisir une approche de chargement 288

Chargeur SQL Aurora 289
Voies de migration 294
Utilisation de PostgreSQL \ copy 296
Ressources supplémentaires 298
IA générative 298
Serveur MCP SQL Aurora d'AWS Labs 298
Pilotage SQL d'Aurora : compétences et pouvoirs 307
Éditeur de requête 312
Conditions préalables 313
Utilisation de l'éditeur de requêtes 313
Éditeurs de requêtes : utilisation JupyterLab avec Aurora DSQL 315
Prise en main 316
Exemple de carnet 318
Suggestions de lecture 318
Sauvegarde et restauration 319
Démarrer avec AWS Backup 319
Restauration de vos sauvegardes 320
Restauration de clusters à une seule région 320
Restauration de clusters multi-régions 320
Surveillance et conformité 320
Ressources supplémentaires 321
Surveillance et journalisation 322
Surveillance avec CloudWatch 322
Observabilité 322
Utilisation 324
Journalisation avec CloudTrail 325
Événements de gestion 326
Événements de données 327
Sécurité 329
AWS politiques gérées 330
AmazonAuroraDSQLFullAccès 330
AmazonAuroraDSQLReadOnlyAccess 332
AmazonAuroraDSQLConsoleFullAccess 332
Aurora DSQLService RolePolicy 334
Mises à jour des politiques 334
Protection des données 342

Chiffrement des données 343
Protection des données dans les régions témoins 345
Certificats SSL/TLS 345
Chiffrement des données 343
Types de clés KMS 352
Chiffrement au repos 353
Utilisation des clés KMS et des clés de données 354
Autoriser votre clé KMS 356
Contexte de chiffrement 358
Surveillance AWS KMS 359
Création d'un cluster chiffré 362
Suppression ou mise à jour d'une clé 364
Considérations 366
Gestion des identités et des accès 367
Public ciblé 367
Authentification par des identités 368
Gestion de l'accès à l'aide de politiques 369
Fonctionnement d'Aurora DSQL avec IAM 371
Exemples de politiques basées sur l'identité 377
Résolution des problèmes 383
Politiques basées sur les ressources 385
Utilisation 386
Créez avec des politiques 387
Ajouter et modifier des politiques 390
Afficher la politique 392
Supprimer la politique 394
Exemples de politiques 395
Blocage de l'accès public 400
Opérations d'API 403
Utilisation d'un rôle lié à un service 406
Autorisations des rôles liés à un service pour Aurora DSQL 406
Créer un rôle lié à un service 407
Modification d'un rôle lié à un service 407
Supprimer un rôle lié à un service 408
Régions prises en charge pour les rôles liés à un service Aurora DSQL 408
Utilisation de clés de condition IAM 408

Création d'un cluster dans une région spécifique 408
Création d'un cluster multi-régions dans des régions spécifiques 409
Création d'un cluster multi-régions avec une région témoin spécifique 410
Intervention en cas d'incidents 411
Validation de conformité 411
Résilience 412
Sauvegarde et restauration 412
Réplication 413
Haute disponibilité 413
Test d'injection de pannes 414
Sécurité de l'infrastructure 415
Gestion des clusters à l'aide de AWS PrivateLink 415
Analyse de la configuration et des vulnérabilités 427
Prévention du cas de figure de l'adjoint désorienté entre services 427
Bonnes pratiques de sécurité 429
Bonnes pratiques de sécurité de détection 429
Bonnes pratiques de sécurité préventive 430
Balisage de ressources 432
Identification de nom 432
Balisage des exigences 432
Balise des notes d'utilisation 433
Considérations 434
Quotas et limites 436
Quotas de cluster 436
Limites de base de données 437
Référence d'API 442
Résolution des problèmes 278
Erreurs de connexion 443
Erreurs d'authentification 444
Erreurs d'autorisation 445
Erreurs SQL 446
Erreurs OCC 446
Connexions SSL/TLS 446
Soumission de commentaires 448
Canaux de feedback 448
Demandes de fonctionnalités efficaces 448

Historique de la documentation 449
..... cdlxvii

Qu'est-ce qu'Amazon Aurora DSQL ?

Amazon Aurora DSQL est un service de base de données relationnelle distribuée sans serveur optimisée pour les charges de travail transactionnelles. Aurora DSQL offre une capacité de mise à l'échelle pratiquement illimitée et ne vous oblige pas à gérer l'infrastructure. L'architecture haute disponibilité active-active assure une disponibilité de 99,99 % dans une seule région et un disponibilité multi-régions de 99,999 %.

Quand utiliser Aurora DSQL

Aurora DSQL est optimisé pour les charges de travail transactionnelles qui tirent parti des transactions ACID et d'un modèle de données relationnel. En raison de son infrastructure sans serveur, Aurora DSQL est idéal pour les modèles d'application des architectures microservice, sans serveur et basées sur les événements. Aurora DSQL étant compatible avec PostgreSQL, vous pouvez utiliser des pilotes, des mappages relationnels objets (), des frameworks et des fonctionnalités SQL courants. ORMs

Aurora DSQL gère automatiquement l'infrastructure du système et met à l'échelle le calcul, les E/S et le stockage en fonction de votre charge de travail. Comme vous n'avez pas de serveurs à approvisionner ni à gérer, vous n'avez pas à vous soucier des durées d'indisponibilité liées à l'approvisionnement, aux correctifs ou aux mises à niveau de l'infrastructure.

Aurora DSQL vous aide à créer et à gérer des applications d'entreprise toujours disponibles à n'importe quelle échelle. La conception sans serveur active-active automatise la reprise après une défaillance, de sorte que vous n'avez pas à vous préoccuper du basculement traditionnel de la base de données. Vos applications bénéficient d'une disponibilité multi-AZ et multi-régions, et vous n'avez pas à vous préoccuper de la cohérence ni des données manquantes éventuelles liées aux basculements.

Principales fonctionnalités d'Aurora DSQL

Les fonctionnalités clés suivantes vous aident à créer une base de données distribuée sans serveur pour prendre en charge vos applications à haute disponibilité :

Architecture distribuée

Aurora DSQL est composé des composants multi-locataires suivants :

- Relais et connectivité

- Calcul et bases de données
- Journal des transactions, contrôle de la simultanéité et isolation
- Stockage

Un plan de contrôle coordonne les composants précédents. Chaque composant assure la redondance entre trois zones de disponibilité (AZs), avec une mise à l'échelle automatique du cluster et une autoréparation en cas de défaillance des composants. Pour plus d'informations sur la manière dont cette architecture prend en charge la haute disponibilité, consultez Résilience dans Amazon Aurora DSQL.

Clusters à une seule région et multi-régions

Les clusters Aurora DSQL offrent les avantages suivants :

- Réplication de données synchrone
- Opérations de lecture cohérentes
- Récupération automatique après des défaillances
- Cohérence des données entre plusieurs AZs régions

En cas de défaillance d'un composant d'infrastructure, Aurora DSQL achemine automatiquement les demandes vers une infrastructure saine sans intervention manuelle. Aurora DSQL fournit des transactions d'atomicité, de cohérence, d'isolement et de durabilité (ACID) avec une cohérence, un isolement des instantanés, une atomicité et une durabilité entre plusieurs AZ et entre plusieurs régions.

Les clusters associés multi-régions offrent la même résilience et la même connectivité que les clusters à une seule région. Mais ils améliorent la disponibilité en proposant deux points de terminaison régionaux, un dans chaque région de cluster associé. Les deux points de terminaison d'un cluster associé présentent une seule base de données logique. Ils sont disponibles pour des opérations de lecture et d'écriture simultanées et assurent une forte cohérence des données. Vous pouvez créer des applications qui s'exécutent dans plusieurs régions en même temps pour des raisons de performances et de résilience, tout en sachant que les lecteurs verront toujours les mêmes données.

Compatibilité avec PostgreSQL

La couche de base de données distribuée (calcul) dans Aurora DSQL est basée sur une version majeure actuelle de PostgreSQL. Vous pouvez vous connecter à Aurora DSQL à l'aide de pilotes et d'outils PostgreSQL courants, tels que `psql`. Aurora DSQL est actuellement compatible avec PostgreSQL version 16 et prend en charge un large éventail de fonctionnalités, d'expressions et

de types de données PostgreSQL. Pour plus d'informations sur les fonctionnalités SQL prises en charge, consultez Compatibilité des fonctionnalités SQL dans Aurora DSQL.

Disponibilité régionale pour Aurora DSQL

Avec Amazon Aurora DSQL, vous pouvez déployer des instances de base de données sur plusieurs Régions AWS pour prendre en charge des applications mondiales et répondre aux exigences de résidence des données. La disponibilité des régions détermine l'endroit où vous pouvez créer et gérer les clusters de bases de données Aurora DSQL. Les administrateurs de bases de données et les architectes d'applications qui doivent concevoir des systèmes de bases de données hautement disponibles et distribués à l'échelle mondiale ont souvent besoin de comprendre la prise en charge de leurs charges de travail par la région. Les cas d'utilisation courants incluent la configuration de la reprise après sinistre entre plusieurs régions, le service aux utilisateurs à partir d'instances de base de données géographiquement plus proches afin de réduire la latence et la conservation de copies de données dans des emplacements spécifiques à des fins de conformité.

Le tableau suivant indique les Régions AWS endroits où Aurora DSQL est actuellement disponible et le point de terminaison de chacun d'entre eux Région AWS.

Nom de la région	Région	Point de terminaison	Protocole
US East (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
		dsql-fips.us-east-2.api.aws	HTTPS
USA Est (Virginie du Nord)	us-east-1	dsql.us-east-1.api.aws	HTTPS
		dsql-fips.us-east-1.api.aws	HTTPS
USA Ouest (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS
		dsql-fips.us-west-2.api.aws	HTTPS
Asie-Pacifique (Melbourne)	ap-southeast-4	dsql.ap-southeast-4.api.aws	HTTPS

Nom de la région	Région	Point de terminaison	Protocole
Asie-Pacifique (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asie-Pacifique (Sydney)	ap-southeast-2	dsql.ap-southeast-2.api.aws	HTTPS
Asie-Pacifique (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Canada (Centre)	ca-central-1	dsql.ca-central-1.api.aws dsql-fips.ca-central-1.api.aws	HTTPS HTTPS
Canada-Ouest (Calgary)	ca-west-1	dsql.ca-west-1.api.aws dsql-fips.ca-west-1.api.aws	HTTPS HTTPS
Europe (Francfort)	eu-central-1	dsql.eu-central-1.api.aws	HTTPS
Europe (Irlande)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europe (Londres)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europe (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS

Disponibilité des clusters multi-régions pour Aurora DSQL

Vous pouvez créer des clusters multirégionaux Aurora DSQL au sein d'ensembles de AWS régions spécifiques. Chaque ensemble de régions regroupe des régions liées géographiquement qui peuvent fonctionner ensemble au sein d'un cluster multi-régions.

Régions US

- USA Est (Virginie du Nord)
- USA Est (Ohio)
- USA Ouest (Oregon)

Régions Asie-Pacifique

- Asie-Pacifique (Osaka)
- Asie-Pacifique (Séoul)
- Asie-Pacifique (Tokyo)

Régions d'Europe

- Europe (Francfort)
- Europe (Irlande)
- Europe (Londres)
- Europe (Paris)

Limitations importantes

Les clusters multi-régions doivent être créés au sein d'un seul ensemble de régions. Par exemple, vous ne pouvez pas créer de cluster qui inclut à la fois les régions USA Est (Virginie du Nord) et Europe (Irlande).

> **Important**
>
> Aurora DSQL ne prend pas actuellement en charge les clusters multi-régions.

Tarification d'Aurora DSQL

Pour plus d'informations sur les coûts, consultez Tarification d'Aurora DSQL.

Quelle est la prochaine étape ?

Pour plus d'informations sur les composants principaux d'Aurora DSQL et pour démarrer avec le service, consultez les rubriques suivantes :

- Mise en route avec Aurora DSQL
- Compatibilité des fonctionnalités SQL dans Aurora DSQL
- Accès à Aurora DSQL avec des clients compatibles avec PostgreSQL
- Aurora DSQL et PostgreSQL

Mise en route avec Aurora DSQL

Amazon Aurora DSQL est une base de données relationnelle distribuée, entièrement gérée et sans serveur optimisée pour les charges de travail transactionnelles. Dans les sections suivantes, vous allez apprendre à créer des clusters Aurora DSQL à région unique ou multirégionale, à vous y connecter et à créer et charger un exemple de schéma. Vous accéderez aux clusters via la AWS console et interagirez éventuellement avec votre base de données à l'aide d'autres clients PostgreSQL. À la fin, vous disposerez d'un cluster Aurora DSQL fonctionnel prêt à être utilisé pour les charges de travail de test ou de production.

Rubriques

- Conditions préalables
- Étape 1 : création d'un cluster à une seule région Aurora DSQL
- Étape 2 : connexion à votre cluster Aurora DSQL
- Étape 3 : exécution d'exemples de commandes SQL dans Aurora DSQL
- Étape 4 (facultatif) : créer un cluster multirégional
- Résolution des problèmes

Conditions préalables

Avant de commencer à utiliser Aurora DSQL, assurez-vous de remplir les prérequis suivants :

- Votre identité IAM doit être autorisée à vous connecter à la console.
- Votre identité IAM doit répondre aux critères suivants :
  - Accès pour effectuer n'importe quelle action sur n'importe quelle ressource de votre Compte AWS
  - `AmazonAuroraDSQLConsoleFullAccess` AWS une politique gérée est jointe.

Étape 1 : création d'un cluster à une seule région Aurora DSQL

L'unité de base d'Aurora DSQL est le cluster, dans lequel vous stockez vos données. Dans cette tâche, vous créez un cluster en une seule Région AWS.

Pour créer un cluster à une seule région dans Aurora DSQL

1. Connectez-vous à la console Aurora AWS Management Console DSQL et ouvrez-la à https://console.aws.amazon.com/dsql l'adresse.
2. Choisissez Créer un cluster, puis Une seule région.
3. (Facultatif) modifiez la valeur de la balise Name par défaut.
4. (Facultatif) Ajoutez des balises supplémentaires pour ce cluster.
5. (Facultatif) Dans Paramètres du cluster, sélectionnez l'une des options suivantes :
   - Sélectionnez Personnaliser les paramètres de chiffrement (avancé) pour choisir ou créer une AWS KMS key. Si vous utilisez une clé gérée par le client, assurez-vous que la politique des clés accorde à Aurora DSQL les autorisations requises. Pour de plus amples informations, veuillez consulter Politique de clé pour une clé gérée par le client.
   - Sélectionnez Activer la protection contre la suppression pour empêcher la suppression de votre cluster. Par défaut, la protection contre la suppression est sélectionnée.
   - Sélectionnez Stratégie basée sur les ressources (avancée) pour spécifier les politiques de contrôle d'accès pour ce cluster.
6. Choisissez Créer un cluster.
7. La console vous renvoie à la page Clusters. Une bannière de notification apparaît pour indiquer que le cluster est en cours de création. Sélectionnez l'ID du cluster pour ouvrir la vue détaillée du cluster.

Étape 2 : connexion à votre cluster Aurora DSQL

Aurora SQL prend en charge plusieurs méthodes de connexion à votre cluster, notamment l'éditeur de requêtes SQL AWS CloudShell, le client PSQL local et d'autres outils compatibles avec PostgreSQL. Au cours de cette étape, vous vous connectez à l'aide de l'éditeur de requêtes SQL Aurora, qui permet de commencer rapidement à interagir avec votre nouveau cluster.

Pour vous connecter à l'aide de l'éditeur de requêtes

1. Dans la console Aurora DSQL (https://console.aws.amazon.com/dsql), ouvrez la page Clusters et vérifiez que la création de votre cluster est terminée et que son statut est Actif.
2. Sélectionnez votre cluster dans la liste ou choisissez l'ID du cluster pour ouvrir la page des détails du cluster.

3. Choisissez Connect with Query editor.
4. Choisissez Connect en tant qu'administrateur pour le cluster qui vient d'être créé.
   - Vous pouvez éventuellement vous connecter avec un rôle personnalisé, voirUtilisation des rôles de base de données et de l'authentification IAM.

Étape 3 : exécution d'exemples de commandes SQL dans Aurora DSQL

Testez votre cluster Aurora DSQL en exécutant des instructions SQL. Après avoir ouvert le cluster dans l'éditeur de requêtes, sélectionnez et exécutez chaque exemple de requête étape par étape.

Exécuter des exemples de commandes SQL dans Aurora DSQL

1. Créez un schéma nommé `test`.

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

2. Créez une table hello_world qui utilise un UUID généré automatiquement comme clé primaire.

```
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. Insérez un exemple de ligne.

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

4. Lisez les valeurs insérées.

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

5. Nettoyage facultatif

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

Étape 4 (facultatif) : créer un cluster multirégional

Si vous souhaitez créer un cluster multi-régions, indiquez les régions suivantes :

Région distante

Il s'agit de la région dans laquelle vous créez un deuxième cluster. Vous créez un deuxième cluster dans cette région et vous le connectez à votre cluster initial. Aurora DSQL réplique toutes les écritures du cluster initial vers le cluster distant. Vous pouvez lire et écrire sur n'importe quel cluster.

Région témoin

Cette région reçoit toutes les données écrites dans le cluster multi-régions. Toutefois, les régions témoins n'hébergent pas de points de terminaison clients et ne fournissent pas d'accès aux données utilisateur. Une fenêtre limitée du journal des transactions chiffré est conservée dans les régions témoins. Ce journal facilite la restauration et soutient le quorum transactionnel en cas d'indisponibilité d'une région.

Utilisez la procédure suivante pour créer un cluster initial, créer un deuxième cluster dans une région différente, puis comparer les deux clusters pour créer un cluster multirégional. Il montre également la réplication d'écriture entre plusieurs régions et les lectures cohérentes à partir des deux points de terminaison régionaux.

Pour créer un cluster multi-régions

1. Connectez-vous à la console Aurora DSQL.
2. Dans le panneau de navigation, choisissez Clusters.
3. Choisissez Créer un cluster, puis Multi-région.
4. (Facultatif) modifiez la valeur de la balise Name par défaut.
5. (Facultatif) Ajoutez des balises supplémentaires pour ce cluster.
6. Dans Paramètres multi-régions, choisissez les options suivantes pour votre cluster initial :
   - Dans Région témoin, choisissez une région. Actuellement, seules les régions basées aux États-Unis sont prises en charge pour les régions témoins dans les clusters multi-régions.
   - (Facultatif) Dans ARN du cluster de région distante, entrez un ARN pour un cluster existant dans une autre région. S'il n'existe aucun cluster pouvant servir de deuxième cluster dans votre cluster multi-régions, terminez la configuration après avoir créé le cluster initial.

7. (Facultatif) Dans Paramètres du cluster, sélectionnez l'une des options suivantes pour votre cluster initial :

   - Sélectionnez Personnaliser les paramètres de chiffrement (avancé) pour choisir ou créer une AWS KMS key. Si vous utilisez une clé gérée par le client, assurez-vous que la politique des clés accorde à Aurora DSQL les autorisations requises. Pour de plus amples informations, veuillez consulter Politique de clé pour une clé gérée par le client.
   - Sélectionnez Activer la protection contre la suppression pour empêcher la suppression de votre cluster. Par défaut, la protection contre la suppression est sélectionnée.
   - Sélectionnez Stratégie basée sur les ressources (avancée) pour spécifier les politiques de contrôle d'accès pour ce cluster.

8. Choisissez Créer un cluster pour créer votre cluster initial. Si vous n'avez pas saisi d'ARN à l'étape précédente, la console affiche la notification Configuration du cluster en attente.
9. Dans la notification Configuration du cluster en attente, choisissez Compléter la configuration du cluster multi-régions. Cette action initie la création d'un deuxième cluster dans une autre région.
10. Choisissez l'une des options suivantes pour votre deuxième cluster :

    - Ajouter un ARN de cluster de région distante : choisissez cette option si un cluster existe et que vous souhaitez qu'il soit le deuxième cluster de votre cluster multi-régions.
    - Créer un cluster dans une autre région : choisissez cette option pour créer un deuxième cluster. Dans Région distante, choisissez la région pour ce deuxième cluster.

11. Choisissez Créer un cluster dans ***your-second-region***, où se *your-second-region* trouve l'emplacement de votre deuxième cluster. La console s'ouvre dans votre deuxième région.
12. (Facultatif) Choisissez les paramètres de cluster pour votre deuxième cluster. Par exemple, vous pouvez choisir une AWS KMS key. Si vous utilisez une clé gérée par le client, assurez-vous que la politique des clés accorde à Aurora DSQL les autorisations requises. Pour de plus amples informations, veuillez consulter Politique de clé pour une clé gérée par le client.
13. Choisissez Créer un cluster pour créer votre deuxième cluster.
14. Choisissez Peer in ***initial-cluster-region***, où se *initial-cluster-region* trouve la région qui héberge le premier cluster que vous avez créé.
15. Lorsque vous y êtes invité, choisissez Confirmer. Cette étape termine la création de votre cluster multi-régions.

Connexion à votre deuxième cluster

1. Ouvrez la console Aurora DSQL et choisissez la région pour votre deuxième cluster.
2. Choisissez Clusters.
3. Sélectionnez la ligne correspondant au deuxième cluster de votre cluster multi-régions.
4. Choisissez Connect with Query editor.
5. Choisissez Connecter en tant qu'administrateur.
6. Créez un exemple de schéma et de table, puis insérez des données en suivant les étapes décrites dansÉtape 3 : exécution d'exemples de commandes SQL dans Aurora DSQL.

Interrogation des données du deuxième cluster à partir de la région hébergeant votre cluster initial

1. Dans la console Aurora DSQL, choisissez la région pour votre cluster initial.
2. Choisissez Clusters.
3. Sélectionnez la ligne correspondant au deuxième cluster de votre cluster multi-régions.
4. Choisissez Connect with Query editor.
5. Choisissez Connecter en tant qu'administrateur.
6. Interrogez les données que vous avez insérées dans le deuxième cluster.

   Example

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

Résolution des problèmes

Consultez la section Dépannage de la documentation d'Aurora DSQL.

Authentification et autorisation pour Aurora DSQL

Aurora DSQL utilise des rôles et des politiques IAM pour l'autorisation des clusters. Vous associez des rôles IAM à des rôles de base de données PostgreSQL pour l'autorisation de base de données. Cette approche combine les avantages d'IAM avec les privilèges de PostgreSQL. Aurora DSQL utilise ces fonctionnalités pour fournir une stratégie d'autorisation et d'accès complète pour votre cluster, votre base de données et vos données.

Gestion de votre cluster à l'aide d'IAM

Pour gérer votre cluster, utilisez IAM pour l'authentification et l'autorisation :

Authentification IAM

Pour authentifier votre identité IAM lorsque vous gérez des clusters Aurora DSQL, vous devez utiliser IAM. Vous pouvez fournir une authentification à l'aide de la AWS Management Console, l'AWS CLI ou le lit AWS SDK.

Autorisation IAM

Pour gérer les clusters Aurora DSQL, accordez l'autorisation à l'aide d'actions IAM pour Aurora DSQL. Par exemple, pour décrire un cluster, assurez-vous que votre identité IAM dispose d'autorisations pour l'action IAM `dsql:GetCluster`, comme dans l'exemple d'action de politique suivant.

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

Pour de plus amples informations, veuillez consulter Utilisation des actions de politique IAM pour gérer les clusters.

Connexion à votre cluster à l'aide d'IAM

Pour connecter votre cluster, utilisez IAM pour l'authentification et l'autorisation :

Authentification IAM

Générez un jeton d'authentification temporaire à l'aide d'une identité IAM avec l'autorisation de se connecter à votre cluster. Pour en savoir plus, consultez Création d'un jeton d'authentification dans Amazon Aurora DSQL.

Autorisation IAM

Accordez les actions de politique IAM suivantes à l'identité IAM que vous utilisez pour établir la connexion au point de terminaison de votre cluster :

- Utilisez `dsql:DbConnectAdmin` si vous utilisez le rôle `admin`. Aurora DSQL crée et gère ce rôle pour vous. L'exemple d'action de politique IAM suivant permet `admin` de se connecter à*`my-cluster`*.

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

- Utilisez `dsql:DbConnect` si vous utilisez un rôle de base de données personnalisé. Vous créez et gérez ce rôle à l'aide des commandes SQL de votre base de données. L'exemple d'action de politique IAM suivant permet à un rôle de base de données personnalisé de *`my-cluster`* se connecter pendant une heure au maximum.

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

Une fois que vous avez établi une connexion, votre rôle est autorisé pour une durée maximale d'une heure pour la connexion.

Interaction avec votre base de données à l'aide des rôles de base de données PostgreSQL et des rôles IAM

PostgreSQL gère les autorisations d'accès aux bases de données en utilisant le concept de rôles. Un rôle peut être considéré comme un utilisateur de base de données ou un groupe d'utilisateurs de

base de données, selon la façon dont le rôle est configuré. Vous créez des rôles PostgreSQL à l'aide de commandes SQL. Pour gérer les autorisations au niveau de la base de données, accordez des autorisations PostgreSQL à vos rôles de base de données PostgreSQL.

Aurora DSQL prend en charge deux types de rôles de base de données : un rôle `admin` et des rôles personnalisés. Aurora DSQL crée automatiquement un rôle `admin` prédéfini pour vous dans votre cluster Aurora DSQL. Vous ne pouvez pas modifier le rôle `admin`. Lorsque vous vous connectez à votre base de données en tant que `admin`, vous pouvez émettre un SQL pour créer de nouveaux rôles au niveau de la base de données à associer à vos rôles IAM. Pour permettre aux rôles IAM de se connecter à votre base de données, associez vos rôles de base de données personnalisés à vos rôles IAM.

Authentification

Utilisez le rôle `admin` pour vous connecter à votre cluster. Après avoir connecté votre base de données, utilisez la commande `AWS IAM GRANT` pour associer un rôle de base de données personnalisé à l'identité IAM autorisée à se connecter au cluster, comme dans l'exemple suivant.

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

Pour en savoir plus, consultez Autoriser les rôles de base de données à se connecter à votre cluster.

Autorisation

Utilisez le rôle `admin` pour vous connecter à votre cluster. Exécutez des commandes SQL pour configurer des rôles de base de données personnalisés et octroyer des autorisations. Pour plus d'informations, consultez Rôles de base de données PostgreSQL et Privilèges PostgreSQL dans la documentation PostgreSQL.

Utilisation des actions de politique IAM avec Aurora DSQL

L'action de politique IAM que vous utilisez dépend du rôle que vous utilisez pour vous connecter à votre cluster : un rôle `admin` ou un rôle de base de données personnalisé. La politique dépend également des actions IAM requises pour ce rôle.

Utilisation des actions de politique IAM pour se connecter aux clusters

Lorsque vous vous connectez à votre cluster avec le rôle de base de données par défaut de `admin`, utilisez une identité IAM autorisée pour effectuer l'action de politique IAM suivante.

```
"dsql:DbConnectAdmin"
```

Lorsque vous vous connectez à votre cluster avec un rôle de base de données personnalisé, associez d'abord le rôle IAM au rôle de base de données. L'identité IAM que vous utilisez pour vous connecter à votre cluster doit être autorisée à exécuter l'action de politique IAM suivante.

```
"dsql:DbConnect"
```

Pour plus d'informations sur les rôles de base de données personnalisés, consultez Utilisation des rôles de base de données et de l'authentification IAM.

Utilisation des actions de politique IAM pour gérer les clusters

Lorsque vous gérez vos clusters Aurora DSQL, spécifiez des actions de politique uniquement pour les actions que votre rôle doit effectuer. Par exemple, si votre rôle n'a besoin que d'obtenir des informations sur les clusters, vous pouvez limiter les autorisations du rôle aux seules autorisations `GetCluster` et `ListClusters`, comme dans l'exemple de politique suivant

JSON

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

L'exemple de politique suivant montre toutes les actions de politique IAM disponibles pour la gestion des clusters.

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" : "*"
    }
  ]
}
```

Révocation d'une autorisation à l'aide d'IAM et de PostgreSQL

Vous pouvez révoquer les autorisations permettant à vos rôles IAM d'accéder à vos rôles au niveau de la base de données :

Révocation de l'autorisation d'administrateur pour se connecter aux clusters

Pour révoquer l'autorisation de connexion à votre cluster avec le rôle `admin`, révoquez l'accès de l'identité IAM à `dsql:DbConnectAdmin`. Modifiez la politique IAM ou détachez la politique de l'identité.

Après avoir révoqué l'autorisation de connexion associée à l'identité IAM, Aurora DSQL rejette toutes les nouvelles tentatives de connexion à partir de cette identité IAM. Toute connexion active utilisant l'identité IAM peut rester autorisée pendant toute la durée de la connexion. Pour plus d'informations sur les durées de connexion, consultez Quotas et limites.

Révocation de l'autorisation d'un rôle personnalisé pour se connecter aux clusters

Pour révoquer l'accès aux rôles de base de données autres `admin`, révoquez l'accès de l'identité IAM à `dsql:DbConnect`. Modifiez la politique IAM ou détachez la politique de l'identité.

Vous pouvez également supprimer l'association entre le rôle de base de données et IAM à l'aide de la commande `AWS IAM REVOKE` dans votre base de données. Pour plus d'informations sur la révocation de l'accès à des rôles de base de données, consultez Révocation de l'autorisation d'une base de données à partir d'un rôle IAM.

Vous ne pouvez pas gérer les autorisations associées au rôle de base de données `admin` prédéfini. Pour savoir comment gérer les autorisations pour les rôles de base de données personnalisés, consultez Privilèges PostgreSQL. Les modifications apportées aux privilèges prennent effet lors de la transaction suivante, après qu'Aurora DSQL a validé la transaction de modification.

Création d'un jeton d'authentification dans Amazon Aurora DSQL

Pour vous connecter à Amazon Aurora DSQL avec un client SQL, générez un jeton d'authentification à utiliser comme mot de passe. Ce jeton est utilisé uniquement pour l'authentification de la connexion. Une fois la connexion établie, la connexion reste valide même si le jeton d'authentification expire.

Si vous créez un jeton d'authentification à l'aide de la AWS AWS CLI console SDKs, le jeton expire automatiquement dans 15 minutes par défaut. La durée maximale est de 604 800 secondes, soit une semaine. Pour vous reconnecter à Aurora DSQL depuis votre client, vous pouvez utiliser le même jeton d'authentification s'il n'a pas expiré, ou vous pouvez en générer un nouveau.

Pour commencer à générer un jeton, créez une politique IAM et un cluster dans Aurora DSQL. Utilisez ensuite la AWS console ou AWS CLI le AWS SDKs pour générer un jeton.

Vous devez disposer au minimum des autorisations IAM répertoriées dans Connexion à votre cluster à l'aide d'IAM, selon le rôle de base de données que vous utilisez pour vous connecter.

Rubriques

- Utiliser la AWS console pour générer un jeton d'authentification dans Aurora DSQL
- AWS CloudShell À utiliser pour générer un jeton d'authentification dans Aurora DSQL
- Utilisez le AWS CLI pour générer un jeton d'authentification dans Aurora DSQL
- Utilisez le SDKs pour générer un jeton dans Aurora DSQL

Utiliser la AWS console pour générer un jeton d'authentification dans Aurora DSQL

Aurora DSQL authentifie les utilisateurs à l'aide d'un jeton plutôt que d'un mot de passe. Vous pouvez générer le jeton à partir de la console.

Pour créer un jeton d'authentification

1. Connectez-vous à la console Aurora AWS Management Console DSQL et ouvrez-la à https://console.aws.amazon.com/dsql l'adresse.
2. Choisissez l'ID du cluster pour lequel vous souhaitez générer un jeton d'authentification. Si vous n'avez pas encore créé de cluster, suivez les étapes décrites dans Étape 1 : création d'un cluster à une seule région Aurora DSQL ou Étape 4 (facultatif) : créer un cluster multirégional.
3. Choisissez Connecter, puis sélectionnez Obtenir un jeton.
4. Choisissez si vous voulez vous connecter en tant qu'`admin` ou avec un rôle de base de données personnalisé.
5. Copiez le jeton d'authentification généré et utilisez-le pour Accédez à Aurora DSQL à l'aide de clients SQL.

Pour plus d'informations sur les rôles de base de données personnalisés et IAM dans Aurora DSQL, consultez Authentification et autorisation.

AWS CloudShell À utiliser pour générer un jeton d'authentification dans Aurora DSQL

Avant de pouvoir générer un jeton d'authentification à l'aide de AWS CloudShell, assurez-vous de créer un cluster Aurora DSQL.

Pour générer un jeton d'authentification à l'aide de AWS CloudShell

1. Connectez-vous à la console Aurora AWS Management Console DSQL et ouvrez-la à https://console.aws.amazon.com/dsql l'adresse.
2. En bas à gauche de la AWS console, choisissez AWS CloudShell.
3. Exécutez la commande suivante pour générer un jeton d'authentification pour le rôle `admin`. *`us-east-1`*Remplacez-le par votre région et *`your_cluster_endpoint`* par le point de terminaison de votre propre cluster.

> **Note**
>
> Si vous ne vous connectez pas en tant que `admin`, utilisez `generate-db-connect-auth-token` à la place.

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

Si vous rencontrez des erreurs, consultez Résoudre les problèmes liés à IAM et Comment résoudre les erreurs d'accès refusé ou d'opération non autorisée avec une politique IAM ?.

4. Utilisez la commande suivante pour utiliser `psql` afin de démarrer une connexion à votre cluster.

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

5. Vous devriez être invité à saisir un mot de passe. Copiez le jeton que vous avez généré et assurez-vous de ne pas inclure d'espaces ni de caractères supplémentaires. Collez-le dans le formulaire d'invite `psql` suivant.

```
Password for user admin:
```

6. Appuyez sur Entrée. Vous devriez obtenir une invite PostgreSQL.

```
postgres=>
```

Si vous obtenez une erreur d'accès refusé, assurez-vous que votre identité IAM dispose de l'autorisation `dsql:DbConnectAdmin`. Si vous êtes autorisé et que vous continuez à recevoir des erreurs d'accès refusé, consultez Résoudre les problèmes liés à IAM et Comment résoudre les erreurs d'accès refusé ou d'opération non autorisée avec une politique IAM ?.

Pour plus d'informations sur les rôles de base de données personnalisés et IAM dans Aurora DSQL, consultez Authentification et autorisation.

Utilisez le AWS CLI pour générer un jeton d'authentification dans Aurora DSQL

Lorsque votre cluster est `ACTIVE`, vous pouvez générer un jeton d'authentification sur l'interface de ligne de commande (CLI) à l'aide de la commande `aws dsql`. Utilisez l'une des techniques suivantes :

> **Note**
>
> La génération de jetons est une opération locale qui signe la demande à l'aide de vos informations d'identification IAM actuelles. Il ne contacte pas AWS pour valider les informations d'identification. Si vos informations d'identification sont expirées ou non valides, la génération du jeton réussit tout de même, mais la tentative de connexion échoue. Assurez-vous que vos informations d'identification IAM sont valides avant de générer un jeton.

- Si vous vous connectez avec le rôle `admin`, utilisez l'option `generate-db-connect-admin-auth-token`.
- Si vous vous connectez avec le rôle de base de données personnalisé, utilisez l'option `generate-db-connect-auth-token`.

L'exemple suivant utilise les attributs suivants pour générer un jeton d'authentification pour le rôle `admin`.

- *`your_cluster_endpoint`*— Le point de terminaison du cluster. Il suit le format *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, comme dans l'exemple `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*— Le Région AWS, tel que `us-east-2` ou`us-east-1`.

Les exemples suivants définissent le délai d'expiration du jeton dans 3 600 secondes (1 heure).

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
```

Utilisez le SDKs pour générer un jeton dans Aurora DSQL

Vous pouvez générer un jeton d'authentification pour votre cluster lorsqu'il est à l'état `ACTIVE`. L'exemple de kits SDK utilise les attributs suivants pour générer un jeton d'authentification pour le rôle `admin` :

- *`your_cluster_endpoint`*(ou*`yourClusterEndpoint`*) : point de terminaison de votre cluster Aurora DSQL. Le format de dénomination est *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, comme dans l'exemple `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*(ou*`RegionEndpoint`*) — L' Région AWS endroit dans lequel se trouve votre cluster, tel que `us-east-2` ou`us-east-1`.

Python SDK

> **Tip**
>
> AWS recommande d'utiliser leConnecteur SQL Aurora pour Python, qui gère automatiquement la génération de jetons.

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`generate_db_connect_admin_auth_token`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`GenerateDBConnectAdminAuthToken`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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 recommande d'utiliser leConnecteurs SQL Aurora pour Node.js, qui gère automatiquement la génération de jetons.

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`getDbConnectAdminAuthToken`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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 recommande d'utiliser leConnexion aux clusters Aurora DSQL à l'aide d'un connecteur JDBC, qui gère automatiquement la génération de jetons.

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`generateDbConnectAdminAuthToken`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`db_connect_admin_auth_token`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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 recommande d'utiliser leConnexion aux clusters SQL Aurora à l'aide d'un connecteur Ruby, qui gère automatiquement la génération de jetons.

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`generate_db_connect_admin_auth_token`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`generateDbConnectAdminAuthToken`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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 recommande d'utiliser leConnexion aux clusters SQL Aurora à l'aide d'un connecteur .NET, qui gère automatiquement la génération de jetons.

> **Note**
>
> Le kit SDK officiel pour .NET n'inclut pas d'appel d'API intégré pour générer un jeton d'authentification pour Aurora DSQL. À la place, vous devez utiliser `DSQLAuthTokenGenerator`, qui est une classe d'utilitaire. L'exemple de code suivant décrit comment générer le jeton d'authentification pour .NET.

Vous pouvez générer le jeton des manières suivantes :

- Si vous vous connectez avec le `admin` rôle, utilisez`DbConnectAdmin`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`DbConnect`.

L'exemple suivant utilise la classe d'utilitaire `DSQLAuthTokenGenerator` pour générer le jeton d'authentification pour un utilisateur avec le rôle `admin`. Remplacez-le *`insert-dsql-cluster-endpoint`* par le point de terminaison de votre cluster.

```
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 recommande d'utiliser leConnexion aux clusters SQL Aurora à l'aide d'un connecteur Go, qui gère automatiquement la génération de jetons.

Le AWS SDK pour Go v2 fournit une méthode intégrée pour générer des jetons d'authentification dans `github.com/aws/aws-sdk-go-v2/feature/dsql/auth`le package.

- Si vous vous connectez avec le `admin` rôle, utilisez`auth.GenerateDBConnectAdminAuthToken`.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez`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)
}
```

Utilisation des rôles de base de données et de l'authentification IAM

Aurora DSQL prend en charge l'authentification à l'aide des rôles IAM et des utilisateurs IAM. Vous pouvez utiliser l'une ou l'autre de ces méthodes pour authentifier les bases de données Aurora DSQL.

Rôles IAM

Un rôle IAM est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques mais qui n'est pas associée à une personne spécifique. L'utilisation des rôles IAM fournit des informations d'identification de sécurité temporaires. Vous pouvez temporairement endosser un rôle IAM selon plusieurs manières :

- En changeant de rôle dans AWS Management Console
- En appelant une opération d' AWS API AWS CLI or
- En utilisant une URL personnalisée

Après avoir endossé un rôle, vous pouvez accéder à Aurora DSQL à l'aide des informations d'identification temporaires du rôle. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Identités IAM dans le Guide de l'utilisateur IAM.

Utilisateurs IAM

Un utilisateur IAM est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques et qui est associée à une seule personne ou à une seule application. Les utilisateurs IAM disposent d'informations d'identification à long terme telles que des mots de passe et des clés d'accès qui peuvent être utilisés pour accéder à Aurora DSQL.

> **Note**
>
> Pour exécuter des commandes SQL avec l'authentification IAM, vous pouvez utiliser le rôle IAM ARNs ou l'utilisateur IAM ARNs dans les exemples ci-dessous.

Autoriser les rôles de base de données à se connecter à votre cluster

Créez un rôle IAM et accordez l'autorisation de connexion avec l'action de politique IAM : `dsql:DbConnect`.

La politique IAM doit également accorder l'autorisation d'accéder aux ressources du cluster. Utilisez un caractère générique (`*`) ou suivez les instructions dans Utilisation des clés de condition IAM avec Amazon Aurora DSQL.

Autoriser les rôles de la base de données à utiliser SQL dans votre base de données

Vous devez utiliser un rôle IAM autorisé pour vous connecter à votre cluster.

1. Connectez-vous à votre cluster Aurora DSQL à l'aide d'un utilitaire SQL.

Utilisez le rôle de base de données `admin` avec une identité IAM autorisée pour l'action IAM `dsql:DbConnectAdmin` pour vous connecter à votre cluster.

2. Créez un nouveau rôle de base de données, en veillant à spécifier l'option `WITH LOGIN`.

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

3. Associez le rôle de base de données à l'ARN du rôle IAM.

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

4. Accordez des autorisations au niveau de la base de données au rôle de base de données

   Les exemples suivants utilisent la commande `GRANT` pour fournir une autorisation au sein de la base de données.

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

Pour plus d'informations, consultez PostgreSQL GRANT et Privilèges PostgreSQL dans la documentation PostgreSQL.

Affichage des mappages de rôles entre IAM et la base de données

Pour afficher les mappages entre les rôles IAM et les rôles de base de données, interrogez la table système `sys.iam_pg_role_mappings`.

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

Exemple de sortie :

```
 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)
```

Ce tableau présente tous les mappages entre les rôles IAM (identifiés par leur ARN) et les rôles de base de données PostgreSQL.

Révocation de l'autorisation d'une base de données à partir d'un rôle IAM

Pour révoquer l'autorisation de base de données, utilisez l'opération `AWS IAM REVOKE`.

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

Pour plus d'informations sur la révocation de l'autorisation, consultez Révocation d'une autorisation à l'aide d'IAM et de PostgreSQL.

Aurora DSQL et PostgreSQL

Aurora DSQL est une base de données relationnelle distribuée compatible avec PostgreSQL conçue pour les charges de travail transactionnelles. Aurora DSQL utilise les principaux composants de PostgreSQL tels que l'analyseur, le planificateur, l'optimiseur et le système de types.

La conception d'Aurora DSQL garantit que toutes les syntaxes PostgreSQL prises en charge fournissent un comportement compatible et produisent des résultats de requête identiques. Par exemple, Aurora DSQL fournit des conversions de type, des opérations arithmétiques, ainsi qu'une précision et une échelle numériques identiques à celles de PostgreSQL. Tout écart est documenté.

Aurora DSQL introduit également des fonctionnalités avancées telles que le contrôle de simultanéité optimisé et la gestion distribuée des schémas. Grâce à ces fonctionnalités, vous pouvez utiliser les outils habituels de PostgreSQL tout en bénéficiant des performances et de l'évolutivité d'applications distribuées modernes, natives du cloud.

Points forts de la compatibilité avec PostgreSQL

Aurora DSQL est actuellement basé sur la version 16 de PostgreSQL. Les principaux points saillants sont les suivants :

Protocole filaire

Aurora DSQL utilise le protocole filaire PostgreSQL v3 standard. Cela permet l'intégration avec les clients, pilotes et outils PostgreSQL standard. Par exemple, Aurora DSQL est compatible avec `psql`, `pgjdbc` et `psycopg`.

Syntaxe SQL

Aurora DSQL prend en charge un large éventail d'expressions et de fonctions PostgreSQL standard couramment utilisées dans les charges de travail transactionnelles. Les expressions SQL prises en charge produisent des résultats identiques à ceux de PostgreSQL, notamment :

- Gestion des valeurs nulles
- Comportement de tri
- Mise à l'échelle et précision pour les opérations numériques
- Équivalence pour les opérations de chaîne

Pour de plus amples informations, veuillez consulter Compatibilité des fonctionnalités SQL dans Aurora DSQL.

Gestion des transactions

Aurora DSQL préserve les principales caractéristiques de PostgreSQL, telles que les transactions ACID et un niveau d'isolement équivalent à PostgreSQL Repeatable Read. Pour de plus amples informations, veuillez consulter Contrôle de simultanéité dans Aurora DSQL.

Avantages de l'architecture distribuée

La conception distribuée et sans partage d'Aurora DSQL offre des avantages en termes de performances et d'évolutivité par rapport aux bases de données à nœud unique traditionnelles. Les principales fonctionnalités sont les suivantes :

Contrôle de simultanéité optimiste (OCC)

Aurora DSQL utilise un modèle de contrôle de simultanéité optimiste. Cette approche sans verrouillage empêche les transactions de se bloquer les unes les autres, élimine les blocages et permet une exécution parallèle à haut débit. Ces fonctionnalités rendent Aurora DSQL particulièrement utile pour les applications nécessitant des performances constantes à grande échelle. Pour obtenir plus d'exemples, consultez Contrôle de simultanéité dans Aurora DSQL.

Opérations DDL asynchrones

Aurora DSQL exécute les opérations DDL de manière asynchrone, ce qui permet des lectures et des écritures ininterrompues lors des modifications du schéma. Son architecture distribuée permet à Aurora DSQL d'effectuer les actions suivantes :

- Exécuter les opérations DDL en tant que tâches de fond, afin de minimiser les perturbations.
- Coordonner les modifications du catalogue sous forme de transactions distribuées hautement cohérentes. Cela garantit une visibilité atomique sur tous les nœuds, même en cas de défaillance ou d'opérations simultanées.
- Opérez de manière entièrement distribuée et autonome sur plusieurs zones de disponibilité grâce à des couches de calcul et de stockage découplées.

Pour en savoir plus sur l'utilisation de la commande EXPLAIN dans PostgreSQL, consultez. Transactions DDL et distribuées dans Aurora DSQL

Compatibilité des fonctionnalités SQL dans Aurora DSQL

Dans les sections suivantes, découvrez la prise en charge par Aurora DSQL des types de données et des commandes SQL de PostgreSQL.

Rubriques

- Types de données pris en charge dans Aurora DSQL
- SQL pris en charge pour Aurora DSQL
- Sous-ensembles de commandes SQL pris en charge dans Aurora DSQL
- Migration de PostgreSQL vers Aurora DSQL

Types de données pris en charge dans Aurora DSQL

Aurora DSQL prend en charge un sous-ensemble des types PostgreSQL courants.

Rubriques

- Types de données numériques
- Types de données de caractères
- Types de données de date et d'heure
- Types de données divers
- Types de données d'exécution des requêtes

Types de données numériques

Aurora DSQL supporte les types de données numériques PostgreSQL suivants.

Nom	les alias ;	Plage et précision	Taille de stockage	Prise en charge de l'index
`smallint`	`int2`	-32768 à +32767	2 octets	Oui
`integer`	`int`, `int4`	-2147483648 à +2147483647	4 octets	Oui

Nom	les alias ;	Plage et précision	Taille de stockage	Prise en charge de l'index
`bigint`	`int8`	-9223372036854775808 à +9223372036854775807	8 octets	Oui
`real`	`float4`	Précision à 6 chiffres décimaux	4 octets	Oui
`double precision`	`float8`	Précision à 15 chiffres décimaux	8 octets	Oui

`numeric [ (`*p*`, `*s*`) ]`	`decimal [ (`*p*`, `*s*`) ]`
	
	`dec[ (`*p*`,`*s*`)]`
		Chiffre exact dont la précision peut être choisie. La précision maximale est de 38 et l'échelle maximale est de 37[1]. La valeur par défaut est `numeric (18,6)`.	8 octets + 2 octets par chiffre de précision. La taille maximale est de 27 octets.	Oui

[1] – Si vous ne spécifiez pas explicitement une taille lorsque vous exécutez `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, Aurora DSQL applique les valeurs par défaut. Aurora DSQL applique des limites lorsque vous exécutez des instructions `INSERT` ou `UPDATE`.

Types de données de caractères

Aurora DSQL prend en charge les types de données de caractères PostgreSQL suivants.

Nom	les alias ;	Description	Limites d'Aurora DSQL	Taille de stockage	Prise en charge de l'index
`character [ (`*n*`) ]`	`char [ (`*n*`) ]`	Chaîne de caractères de longueur fixe	4 096 octets[1]	Variable jusqu'à 4 100 octets	Oui

Nom	les alias ;	Description	Limites d'Aurora DSQL	Taille de stockage	Prise en charge de l'index
`character varying [ (n) ]`	`varchar [ (n) ]`	Chaîne de caractères de longueur variable	65 535 octets[1]	Variable jusqu'à 65 539 octets	Oui
`bpchar [ (n) ]`		Si la longueur est fixe, il s'agit d'un alias pour `char`. Si la longueur est variable, il s'agit d'un alias pour `varchar`, où les espaces de fin sont sémantiquement insignifiants.	4 096 octets[1]	Variable jusqu'à 4 100 octets	Oui
`text`		Chaîne de caractères de longueur variable	1 MiO[1]	Variable jusqu'à 1 MiO	Oui

[1] – Si vous ne spécifiez pas explicitement une taille lorsque vous exécutez `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, Aurora DSQL applique les valeurs par défaut. Aurora DSQL applique des limites lorsque vous exécutez des instructions `INSERT` ou `UPDATE`.

Types de données de date et d'heure

Aurora DSQL prend en charge les types de données de date et d'heure PostgreSQL suivants.

Nom	les alias ;	Description	Range	Résolution	Taille de stocka	Prise en charge de l'index
`date`		Date calendaire (année, mois, jour)	4713 BC — 5874897 AD	1 jour	4 octet	Oui
`time [ (p) ] [ without time zone ]`	`time`	Heure du jour, sans fuseau horaire	0 – 1	1 microseconde	8 octet	Oui
`time [ (p) ] with time zone`	`time`	heure du jour, avec le fuseau horaire	00:00:00+1559 – 24:00:00 –1559	1 microseconde	12 octe	Non
`timestamp [ (p) ] [ without time zone ]`		Date et heure, sans fuseau horaire	4713 BC — 294276 AD	1 microseconde	8 octet	Oui
`timestamp [ (p) ] with time zone`	`timestz`	Date et heure, avec le fuseau horaire	4713 BC — 294276 AD	1 microseconde	8 octet	Oui
`interval [ fields ] [ (p) ]`		Durée	-178000000 ans — 178000000 ans	1 microseconde	16 octe	Non

Types de données divers

Aurora DSQL prend en charge les types de données PostgreSQL divers suivants.

Nom	les alias ;	Description	Limites d'Aurora DSQL	Taille de stockage	Prise en charge de l'index
`boolean`	`bool`	Booléen logique (true/ false)		1 octet	Oui
`bytea`		Données binaires (« tableau d'octets »)	1 MiO[1]	Variable jusqu'à la limite de 1 MiO	Non
`UUID`		Identifiant unique et universel		16 octets	Oui

[1] – Si vous ne spécifiez pas explicitement une taille lorsque vous exécutez `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, Aurora DSQL applique les valeurs par défaut. Aurora DSQL applique des limites lorsque vous exécutez des instructions `INSERT` ou `UPDATE`.

Types de données d'exécution des requêtes

Les types de données d'exécution des requêtes sont des types de données internes utilisés au moment de l'exécution des requêtes. Ces types sont distincts des types compatibles avec PostgreSQL tels que `varchar` et `integer` que vous définissez dans votre schéma. Ces types sont plutôt des représentations d'exécution qu'Aurora DSQL utilise lors du traitement d'une requête.

Les types de données suivants sont pris en charge uniquement pendant l'exécution des requêtes :

Type de tableau

Aurora DSQL prend en charge les tableaux des types de données pris en charge. Par exemple, vous pouvez avoir un tableau d'entiers. La fonction `string_to_array` divise une chaîne en un

tableau de style PostgreSQL avec la virgule de séparation (`,`), comme indiqué dans l'exemple suivant. Vous pouvez utiliser des tableaux dans des expressions, des sorties de fonctions ou des calculs temporaires lors de l'exécution de requêtes.

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

La fonction renvoie une réponse similaire à la réponse suivante :

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

type d'inet

Le type de données représente IPv4 les adresses IPv6 d'hôtes et leurs sous-réseaux. Ce type est utile pour analyser des journaux, filtrer sur des sous-réseaux IP ou effectuer des calculs réseau dans le cadre d'une requête. Pour plus d'informations, consultez inet dans la documentation de PostgreSQL.

Fonctions d'exécution JSON

Aurora DSQL prend en charge JSON et JSONB en tant que types de données d'exécution pour le traitement des requêtes. Stockez les données JSON sous forme de `text` colonnes et convertissez-les en JSON lors de l'exécution de la requête afin d'utiliser les fonctions et les opérateurs JSON de PostgreSQL.

Aurora DSQL prend en charge la plupart des fonctions JSON de PostgreSQL de la section 9.1.6 Fonctions et opérateurs JSON avec un comportement identique.

Les fonctions qui renvoient des types JSON ou JSONB peuvent nécessiter une conversion `text` supplémentaire pour un affichage correct.

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

La fonction renvoie une réponse similaire à la réponse suivante :

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

```
(1 row)
```

SQL pris en charge pour Aurora DSQL

Aurora DSQL prend en charge un large éventail de fonctionnalités SQL PostgreSQL de base. Dans les sections suivantes, vous trouverez des informations sur la prise en charge générale des expressions de PostgreSQL. Cette liste n'est pas exhaustive.

Commande **SELECT**

Aurora DSQL prend en charge les clauses suivantes de la commande `SELECT`.

Clause principale	Clauses prises en charge
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	
`WITH` (expressions de table communes)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`

Clause principale	Clauses prises en charge
`EXCEPT`	`ALL`
`OVER`	`RANK ()`, `PARTITION BY`
`FOR UPDATE`	

Langage de définition de données (DDL)

Aurora DSQL prend en charge les commandes DDL PostgreSQL suivantes.

Commande	Clause principale	Clauses prises en charge
`CREATE`	`TABLE`	Pour plus d'informations sur la syntaxe de la commande `CREATE TABLE` prise en charge, consultez `CREATE TABLE`.
`ALTER`	`TABLE`	Pour plus d'informations sur la syntaxe de la commande `ALTER TABLE` prise en charge, consultez `ALTER TABLE`.
`DROP`	`TABLE`	
`CREATE`	`[UNIQUE] INDEX ASYNC`	Vous pouvez utiliser cette commande avec les paramètres suivants : `ON`, `NULLS FIRST`, `NULLS LAST`. Pour plus d'informations sur la syntaxe de la commande `CREATE INDEX ASYNC` prise en charge, consultez Index asynchrones dans Aurora DSQL.
`DROP`	`INDEX`	
`CREATE`	`VIEW`	Pour plus d'informations sur la syntaxe de la commande `CREATE VIEW` prise en charge, consultez `CREATE VIEW`.

Commande	Clause principale	Clauses prises en charge
ALTER	VIEW	Pour plus d'informations sur la syntaxe de la commande ALTER VIEW prise en charge, consultez ALTER VIEW.
DROP	VIEW	Pour plus d'informations sur la syntaxe de la commande DROP VIEW prise en charge, consultez DROP VIEW.
CREATE	SEQUENCE	Pour plus d'informations sur la syntaxe de la commande CREATE SEQUENCE prise en charge, consultez CREATE SEQUENCE.
ALTER	SEQUENCE	Pour plus d'informations sur la syntaxe de la commande ALTER SEQUENCE prise en charge, consultez ALTER SEQUENCE.
DROP	SEQUENCE	Pour plus d'informations sur la syntaxe de la commande DROP SEQUENCE prise en charge, consultez DROP SEQUENCE.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Langage de manipulation de données (DML)

Aurora DSQL prend en charge les commandes DML PostgreSQL suivantes.

Commande	Clause principale	Clauses prises en charge
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT)

Commande	Clause principale	Clauses prises en charge
		`FROM, WITH`
`DELETE`	`FROM`	`USING`, `WHERE`

Langage de contrôle des données (DCL)

Aurora DSQL prend en charge les commandes DCL PostgreSQL suivantes.

Commande	Clauses prises en charge
`GRANT`	`ON`, `TO`
`REVOKE`	`ON`, `FROM`, `CASCADE`, `RESTRICT`

Langage de contrôle des transactions (TCL)

Aurora DSQL prend en charge les commandes TCL PostgreSQL suivantes.

Commande	Clauses prises en charge	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`

Commande	Clauses prises en charge	Alias
	[AND NO CHAIN]	

Commandes d'utilitaire

Aurora DSQL prend en charge les commandes d'utilitaire PostgreSQL suivantes.

- EXPLAIN
- ANALYZE (nom de relation uniquement)

Sous-ensembles de commandes SQL pris en charge dans Aurora DSQL

Cette section fournit des informations détaillées sur les commandes SQL prises en charge, en se concentrant sur les commandes comportant de nombreux ensembles de paramètres et sous-commandes. Par exemple, CREATE TABLE dans PostgreSQL propose de nombreuses clauses et paramètres, dont un sous-ensemble est pris en charge par Aurora DSQL. Cette section décrit les sous-ensembles de commandes SQL courantes pris en charge à l'aide d'éléments de syntaxe PostgreSQL courants pris en charge par Aurora DSQL.

Rubriques

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

CREATE TABLE

CREATE TABLE définit une nouvelle table.

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

Colonnes d'identité

> **Note**
>
> Lorsque vous utilisez des colonnes d'identité, la valeur du cache doit être soigneusement prise en compte. Pour plus d'informations, consultez la légende Important sur la `CREATE SEQUENCE` page.
>
> Pour obtenir des conseils sur la meilleure façon d'utiliser les colonnes d'identité en fonction des modèles de charge de travail, voirUtilisation de séquences et de colonnes d'identité.

La GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY (*sequence_options*) clause crée la colonne en tant que colonne d'identité. Une séquence implicite y sera attachée et dans les lignes nouvellement insérées, la colonne contiendra automatiquement les valeurs de la séquence qui lui est attribuée. Une telle colonne l'est implicitementNOT NULL.

Les clauses ALWAYS et BY DEFAULT déterminent la manière dont les valeurs spécifiées explicitement par l'utilisateur sont traitées dans UPDATE les commandes INSERT et.

Dans une INSERT commande, si elle ALWAYS est sélectionnée, une valeur spécifiée par l'utilisateur n'est acceptée que si l'INSERTinstruction le préciseOVERRIDING SYSTEM VALUE. Si cette option BY DEFAULT est sélectionnée, la valeur spécifiée par l'utilisateur est prioritaire.

Dans une UPDATE commande, si elle ALWAYS est sélectionnée, toute mise à jour de la colonne vers une valeur autre que DEFAULT sera rejetée. Si cette option BY DEFAULT est sélectionnée, la colonne peut être mise à jour normalement. (Il n'y a aucune OVERRIDING clause pour la UPDATE commande.)

La *sequence_options* clause peut être utilisée pour remplacer les paramètres de la séquence. Les options disponibles incluent celles indiquées pourCREATE SEQUENCE, plusSEQUENCE NAME *name*. SansSEQUENCE NAME, le système choisit un nom inutilisé pour la séquence.

ALTER TABLE

ALTER TABLE modifie la définition d'une table.

```
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 }
```

Actions relatives à la colonne d'identité

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

Ces formulaires permettent de déterminer si une colonne est une colonne d'identité ou de modifier l'attribut de génération d'une colonne d'identité existante. Consultez `CREATE TABLE` pour plus de détails. Par exemple`SET DEFAULT`, ces formulaires n'affectent que le comportement `INSERT` des `UPDATE` commandes suivantes ; ils ne modifient pas les lignes déjà présentes dans le tableau.

*`sequence_option`*Il s'agit d'une option prise en charge par `ALTER SEQUENCE` tel que`INCREMENT BY`. Ces formulaires modifient la séquence qui sous-tend une colonne d'identité existante.

`DROP IDENTITY [ IF EXISTS ]`

Ce formulaire supprime la propriété d'identité d'une colonne. Si elle `DROP IDENTITY IF EXISTS` est spécifiée et que la colonne n'est pas une colonne d'identité, aucune erreur n'est renvoyée. Dans ce cas, un avis est émis à la place.

`CREATE SEQUENCE`

`CREATE SEQUENCE`— définit un nouveau générateur de séquences.

> **Important**
>
> Dans PostgreSQL, la `CACHE` spécification est facultative et la valeur par défaut est 1. Dans un système distribué tel qu'Amazon Aurora DSQL, les opérations de séquence impliquent une coordination, et une taille de cache de 1 peut augmenter la charge de coordination en cas de forte simultanéité. Alors que des valeurs de cache plus importantes permettent de servir des numéros de séquence à partir de plages préallouées localement, ce qui améliore le débit, les valeurs réservées non utilisées peuvent être perdues, ce qui rend les écarts et les effets d'ordre plus visibles. Étant donné que la sensibilité des applications à l'ordre d'allocation diffère de celle du débit, Amazon Aurora DSQL doit `CACHE` être spécifiée de manière explicite et prend actuellement en charge `CACHE = 1` ou`CACHE >= 65536`, en établissant une distinction claire entre un comportement d'allocation proche de la génération strictement séquentielle et une allocation optimisée pour des charges de travail hautement simultanées.

Lorsque l'`CACHE >= 65536`unicité des valeurs de séquence reste garantie, mais qu'elles peuvent ne pas être générées dans un ordre strictement croissant d'une session à l'autre, des écarts peuvent survenir, en particulier lorsque les valeurs mises en cache ne sont pas entièrement consommées. Ces caractéristiques sont cohérentes avec la sémantique de PostgreSQL pour les séquences mises en cache utilisées simultanément, où les deux systèmes garantissent des valeurs distinctes mais ne garantissent pas un ordre strictement séquentiel entre les sessions.
Au cours d'une seule session client, les valeurs de séquence peuvent ne pas toujours apparaître strictement croissantes, en particulier en dehors des transactions explicites.
Ce comportement est similaire à celui des déploiements PostgreSQL qui utilisent le regroupement de connexions. Un comportement d'allocation plus proche de celui d'un environnement PostgreSQL mono-session peut être obtenu en `CACHE = 1` utilisant ou en obtenant des valeurs de séquence dans le cadre de transactions explicites.
Avec`CACHE = 1`, l'allocation de séquences suit le comportement des séquences non mises en cache de PostgreSQL.
Pour obtenir des conseils sur la meilleure façon d'utiliser les séquences basées sur les modèles de charge de travail, voirUtilisation de séquences et de colonnes d'identité.

Syntaxe prise en charge

```
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`crée un nouveau générateur de numéros de séquence. Cela implique la création et l'initialisation d'une nouvelle table spéciale à une seule ligne portant le nom. *`name`* Le générateur appartiendra à l'utilisateur qui émet la commande.

Si un nom de schéma est donné, la séquence est créée dans le schéma spécifié. Dans le cas contraire, il est créé dans le schéma actuel. Le nom de séquence doit être distinct du nom de toute autre relation (table, séquence, index, vue, vue matérialisée ou table étrangère) dans le même schéma.

Une fois qu'une séquence est créée, vous utilisez les fonctions `nextval`, `currval`, et `setval` pour opérer sur la séquence. Ces fonctions sont documentées dans Fonctions de manipulation de séquence.

Bien que vous ne puissiez pas mettre à jour une séquence directement, vous pouvez utiliser une requête telle que :

```
SELECT * FROM name;
```

pour examiner certains paramètres et l'état actuel d'une séquence. En particulier, le `last_value` champ de la séquence indique la dernière valeur allouée par une session. (Bien entendu, cette valeur peut être obsolète au moment de son impression, si d'autres sessions effectuent activement des `nextval` appels.) D'autres paramètres tels que *`increment`* et *`maxvalue`* peuvent être observés dans la `pg_sequences` vue.

Parameters

`IF NOT EXISTS`

Ne renvoie pas d'erreur si une relation portant le même nom existe déjà. Un avis est émis dans ce cas. Notez qu'il n'y a aucune garantie que la relation existante ressemble à la séquence qui aurait été créée ; ce n'est peut-être même pas une séquence.

`name`

Nom (éventuellement qualifié par schéma) de la séquence à créer.

`data_type`

La clause facultative `AS` *`data_type`* spécifie le type de données de la séquence. Les types valides sont`bigint`. `bigint`est la valeur par défaut. Le type de données détermine les valeurs minimale et maximale par défaut de la séquence.

`increment`

La clause facultative `INCREMENT BY` *`increment`* indique quelle valeur est ajoutée à la valeur de séquence actuelle pour créer une nouvelle valeur. Une valeur positive fera une séquence ascendante, une valeur négative une séquence descendante. La valeur par défaut est 1.

minvalue / **NO MINVALUE**

La clause facultative `MINVALUE` *minvalue* détermine la valeur minimale qu'une séquence peut générer. Si cette clause n'est pas fournie ou `NO MINVALUE` est spécifiée, les valeurs par défaut seront utilisées. La valeur par défaut pour une séquence ascendante est 1. La valeur par défaut pour une séquence descendante est la valeur minimale du type de données.

maxvalue / **NO MAXVALUE**

La clause facultative `MAXVALUE` *maxvalue* détermine la valeur maximale de la séquence. Si cette clause n'est pas fournie ou `NO MAXVALUE` est spécifiée, les valeurs par défaut seront utilisées. La valeur par défaut pour une séquence ascendante est la valeur maximale du type de données. La valeur par défaut pour une séquence descendante est -1.

CYCLE / **NO CYCLE**

L'`CYCLE`option permet à la séquence de s'enrouler lorsque le *maxvalue* ou *minvalue* a été atteint par une séquence ascendante ou descendante respectivement. Si la limite est atteinte, le prochain nombre généré sera respectivement *maxvalue* le *minvalue* ou.

Si `NO CYCLE` cette option est spécifiée, tout appel effectué une `nextval` fois que la séquence a atteint sa valeur maximale renverra une erreur. Si aucun des deux `CYCLE` ou `NO CYCLE` n'est spécifié, `NO CYCLE` c'est la valeur par défaut.

start

La clause facultative `START WITH` *start* permet à la séquence de commencer n'importe où. La valeur de départ par défaut est *minvalue* pour les séquences ascendantes et *maxvalue* pour les séquences descendantes.

cache

La clause `CACHE` *cache* indique le nombre de numéros de séquence à préallouer et à stocker en mémoire pour un accès plus rapide. Les valeurs acceptables pour Aurora `CACHE` DSQL sont 1 ou n'importe quel nombre >= 65536. La valeur minimale est 1 (une seule valeur peut être générée à la fois, ce qui signifie qu'il n'y a pas de cache).

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

`OWNED BY`Cette option permet d'associer la séquence à une colonne de table spécifique, de sorte que si cette colonne (ou le tableau entier) est supprimée, la séquence sera également automatiquement supprimée. La table spécifiée doit avoir le même propriétaire et être dans

le même schéma que la séquence. `OWNED BY NONE`, valeur par défaut, indique qu'il n'existe aucune association de ce type.

Remarques

`DROP SEQUENCE`À utiliser pour supprimer une séquence.

Les séquences étant basées sur l'`bigint`arithmétique, la plage ne peut pas dépasser la plage d'un entier de huit octets (-9223372036854775808 à 9223372036854775807).

Comme `nextval` les `setval` appels ne sont jamais annulés, les objets de séquence ne peuvent pas être utilisés si l'attribution « sans interruption » de numéros de séquence est nécessaire.

Chaque session allouera et mettra en cache les valeurs de séquence successives lors d'un accès à l'objet de séquence et augmentera celles de l'objet de séquence `last_value` en conséquence. Ensuite, les *`cache`* -1 utilisations suivantes de `nextval` cette session renvoient simplement les valeurs préallouées sans toucher à l'objet de séquence. Ainsi, tous les numéros alloués mais non utilisés au cours d'une session seront perdus à la fin de cette session, ce qui entraînera des « trous » dans la séquence.

En outre, bien que plusieurs sessions soient garanties pour allouer des valeurs de séquence distinctes, les valeurs peuvent être générées hors séquence lorsque toutes les sessions sont prises en compte. Par exemple, avec un *`cache`* paramètre de 10, la session A peut réserver les valeurs 1.. 10 et renvoyer `nextval` =1, puis la session B peut réserver les valeurs 11.. 20 et renvoyer `nextval` =11 avant que la session A n'ait généré `nextval` =2. Ainsi, avec un *`cache`* paramètre égal à un, on peut supposer sans risque de se tromper que `nextval` les valeurs sont générées de manière séquentielle ; avec un *`cache`* paramètre supérieur à un, vous devez uniquement supposer que les `nextval` valeurs sont toutes distinctes, et non qu'elles sont générées de manière purement séquentielle. En outre, `last_value` reflétera la dernière valeur réservée par une session, qu'elle ait déjà été renvoyée ou non`nextval`.

Une autre considération est qu'une `setval` exécution sur une telle séquence ne sera pas remarquée par les autres sessions tant qu'elles n'auront pas utilisé les valeurs préallouées qu'elles ont mises en cache.

Exemples

Créez une séquence ascendante appelée`serial`, à partir de 101 :

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

Sélectionnez le numéro suivant dans cette séquence :

```
SELECT nextval('serial');

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

Sélectionnez le numéro suivant dans cette séquence :

```
SELECT nextval('serial');

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

Utilisez cette séquence dans une `INSERT` commande :

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

Réinitialisez la séquence à une valeur spécifique en utilisant `setval` :

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

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

Compatibilité

`CREATE SEQUENCE`est conforme à la norme SQL, avec les exceptions suivantes :

- L'obtention de la valeur suivante s'effectue à l'aide de la `nextval()` fonction plutôt que de l'`NEXT VALUE FOR`expression de la norme.
- La `OWNED BY` clause est une extension PostgreSQL.

ALTER SEQUENCE

`ALTER SEQUENCE`— modifie la définition d'un générateur de séquences.

> **Important**
>
> Lorsque vous utilisez des séquences, la valeur du cache doit être soigneusement prise en compte. Pour plus d'informations, consultez la légende Important sur la `CREATE SEQUENCE` page.
> Pour obtenir des conseils sur la meilleure façon d'utiliser les séquences basées sur les modèles de charge de travail, voirUtilisation de séquences et de colonnes d'identité.

Syntaxe prise en charge

```
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`modifie les paramètres d'un générateur de séquences existant. Tous les paramètres qui ne sont pas spécifiquement définis dans la `ALTER SEQUENCE` commande conservent leurs paramètres antérieurs.

Vous devez être propriétaire de la séquence à utiliser`ALTER SEQUENCE`. Pour modifier le schéma d'une séquence, vous devez également disposer de `CREATE` privilèges sur le nouveau schéma. Pour modifier le propriétaire, vous devez être en mesure d'`SET ROLE`accéder au nouveau rôle propriétaire, et ce rôle doit disposer de `CREATE` privilèges sur le schéma de la séquence. (Ces restrictions font en sorte que le fait de modifier le propriétaire n'a aucun effet que vous ne pourriez pas faire en supprimant et en recréant la séquence. Cependant, un superutilisateur peut de toute façon modifier la propriété de n'importe quelle séquence.)

Parameters

name

Nom (éventuellement qualifié par schéma) d'une séquence à modifier.

IF EXISTS

Ne renvoie pas d'erreur si la séquence n'existe pas. Un avis est émis dans ce cas.

increment

La clause `INCREMENT BY` *`increment`* est facultative. Une valeur positive fera une séquence ascendante, une valeur négative une séquence descendante. Si elle n'est pas spécifiée, l'ancienne valeur d'incrément sera conservée.

minvalue / **NO MINVALUE**

La clause facultative `MINVALUE` *`minvalue`* détermine la valeur minimale qu'une séquence peut générer. Si elle `NO MINVALUE` est spécifiée, la valeur par défaut de 1 et la valeur minimale du type de données pour les séquences ascendantes et descendantes, respectivement, seront utilisées. Si aucune option n'est spécifiée, la valeur minimale actuelle sera maintenue.

maxvalue / **NO MAXVALUE**

La clause facultative `MAXVALUE` *`maxvalue`* détermine la valeur maximale de la séquence. Si `NO MAXVALUE` cette option est spécifiée, les valeurs par défaut de la valeur maximale du type de données et de -1 pour les séquences ascendantes et descendantes, respectivement, seront utilisées. Si aucune option n'est spécifiée, la valeur maximale actuelle sera maintenue.

CYCLE

Le mot `CYCLE` clé facultatif peut être utilisé pour permettre à la séquence de s'enrouler lorsque le *`maxvalue`* ou *`minvalue`* a été atteint par une séquence ascendante ou descendante respectivement. Si la limite est atteinte, le prochain nombre généré sera respectivement *`maxvalue`* le *`minvalue`* ou.

NO CYCLE

Si le mot `NO CYCLE` clé facultatif est spécifié, tout appel effectué `nextval` après que la séquence a atteint sa valeur maximale renverra une erreur. Si aucun `CYCLE` `NO CYCLE` des deux n'est spécifié, le comportement de l'ancien cycle sera conservé.

start

La clause facultative `START WITH` *`start`* modifie la valeur de départ enregistrée de la séquence. Cela n'a aucun effet sur la valeur de séquence actuelle ; cela définit simplement la valeur que `ALTER SEQUENCE RESTART` les futures commandes utiliseront.

restart

La clause facultative `RESTART [ WITH` *`restart`* `]` modifie la valeur actuelle de la séquence. Cela revient à appeler la `setval` fonction avec `is_called` = `false` : la valeur spécifiée sera renvoyée lors du prochain appel de`nextval`. Écrire `RESTART` sans *`restart`* valeur équivaut à fournir la valeur de départ enregistrée par `CREATE SEQUENCE` ou définie pour la dernière fois par`ALTER SEQUENCE START WITH`.

Contrairement à un `setval` appel, une `RESTART` opération sur une séquence est transactionnelle et empêche les transactions simultanées d'obtenir des numéros à partir de la même séquence. Si ce n'est pas le mode de fonctionnement souhaité, `setval` il doit être utilisé.

cache

La clause `CACHE` *`cache`* permet de préallouer les numéros de séquence et de les stocker en mémoire pour un accès plus rapide. La valeur doit être soit 1, soit une valeur >= 65536. Si elle n'est pas spécifiée, l'ancienne valeur du cache sera conservée. Pour plus d'informations sur le comportement du cache, consultez les instructions ci-dessous`CREATE SEQUENCE`.

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

`OWNED BY`Cette option permet d'associer la séquence à une colonne de table spécifique, de sorte que si cette colonne (ou le tableau entier) est supprimée, la séquence sera également automatiquement supprimée. Si elle est spécifiée, cette association remplace toute association précédemment spécifiée pour la séquence. La table spécifiée doit avoir le même propriétaire et être dans le même schéma que la séquence. La spécification `OWNED BY NONE` supprime toute association existante, rendant la séquence « autonome ».

new_owner

Le nom d'utilisateur du nouveau propriétaire de la séquence.

new_name

Le nouveau nom de la séquence.

new_schema

Le nouveau schéma de la séquence.

Remarques

`ALTER SEQUENCE`n'affectera pas immédiatement les `nextval` résultats dans les backends, autres que celui en cours, qui ont des valeurs de séquence préallouées (mises en cache). Ils utiliseront toutes les valeurs mises en cache avant de remarquer les paramètres de génération de séquence modifiés. Le backend actuel sera immédiatement affecté.

`ALTER SEQUENCE`n'affecte pas le `currval` statut de la séquence.

`ALTER SEQUENCE`peut entraîner d'autres transactions pour OCC.

Pour des raisons historiques, il `ALTER TABLE` peut également être utilisé avec des `ALTER TABLE` séquences ; mais les seules variantes autorisées avec des séquences sont équivalentes aux formes présentées ci-dessus.

Exemples

Redémarrer une séquence appelée`serial`, à 105 :

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

Compatibilité

`ALTER SEQUENCE`est conforme à la norme SQL, à l'exception des `SET SCHEMA` clauses`AS`,`START WITH`,`OWNED BY`,`OWNER TO`, et`RENAME TO`, qui sont des extensions PostgreSQL.

DROP SEQUENCE

`DROP SEQUENCE`— supprime une séquence.

Syntaxe prise en charge

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

Description

`DROP SEQUENCE`supprime les générateurs de numéros de séquence. Une séquence ne peut être supprimée que par son propriétaire ou par un superutilisateur.

Parameters

`IF EXISTS`

Ne renvoie pas d'erreur si la séquence n'existe pas. Un avis est émis dans ce cas.

`name`

Le nom (éventuellement qualifié par un schéma) d'une séquence.

`CASCADE`

Supprimez automatiquement les objets qui dépendent de la séquence et, à leur tour, tous les objets qui dépendent de ces objets.

`RESTRICT`

Refusez de supprimer la séquence si des objets en dépendent. Il s'agit de l'option par défaut.

Exemples

Pour supprimer la séquence, procédez comme suit `seq` :

```
DROP SEQUENCE seq;
```

Compatibilité

`DROP SEQUENCE`est conforme à la norme SQL, sauf que la norme ne permet de supprimer qu'une seule séquence par commande, et à l'exception de l'`IF EXISTS`option, qui est une extension PostgreSQL.

`CREATE VIEW`

`CREATE VIEW` définit une nouvelle vue persistante. Aurora DSQL ne prend pas en charge les vues temporaires ; seules les vues permanentes sont prises en charge.

Syntaxe prise en charge

```
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` définit une vue d'une requête. La vue n'est pas matérialisée physiquement. Au lieu de cela, la requête est exécutée chaque fois que la vue est référencée dans une requête.

`CREATE or REPLACE VIEW` est similaire, mais si une vue du même nom existe déjà, elle est remplacée. La nouvelle requête doit générer les mêmes colonnes que celles générées par la requête de vue existante (c'est-à-dire les mêmes noms de colonnes dans le même ordre et avec les mêmes types de données), mais elle peut ajouter des colonnes supplémentaires à la fin de la liste. Les calculs à l'origine des colonnes de sortie peuvent être différents.

Si un nom de schéma est fourni (tel que `CREATE VIEW myschema.myview ...`), la vue est créée dans le schéma spécifié. Sinon, elle est créée dans le schéma en cours.

Le nom de la vue doit être distinct du nom de toute autre relation (table, index, vue) dans le même schéma.

Parameters

`CREATE VIEW` prend en charge divers paramètres de contrôle du comportement des vues pouvant être mises à jour automatiquement.

`RECURSIVE`

Crée une vue récursive. La syntaxe : `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` est équivalente à `CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;`.

Une liste de noms de colonnes de vue doit être spécifiée pour une vue récursive.

`name`

Le nom de la vue à créer, qui peut éventuellement être qualifié de schéma. Une liste de noms de colonnes doit être spécifiée pour une vue récursive.

`column_name`

Liste facultative des noms à utiliser pour les colonnes de la vue. Si elle n'est pas fournie, les noms de colonnes sont déduits de la requête.

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

Cette clause spécifie des paramètres facultatifs pour une vue ; les paramètres suivants sont pris en charge.

- `check_option (enum)` : ce paramètre peut être `local` ou `cascaded` et équivaut à la spécification `WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)` : cela doit être utilisé si la vue est destinée à fournir une sécurité au niveau des lignes. Aurora DSQL ne prend actuellement pas en charge la sécurité au niveau des lignes, mais cette option obligera tout de même à évaluer d'`WHERE`abord les conditions de la vue (et toutes les conditions utilisant des opérateurs marqués comme`LEAKPROOF`).
- `security_invoker (boolean)` : cette option permet de vérifier les relations de base sous-jacentes en fonction des privilèges de l'utilisateur de la vue plutôt que du propriétaire de la vue. Pour plus de détails, consultez les notes ci-dessous.

Toutes les options ci-dessus peuvent être modifiées sur les vues existantes à l'aide de `ALTER VIEW`.

`query`

Une `VALUES` commande `SELECT` ou qui fournira les colonnes et les lignes de la vue.

`WITH [ CASCADED | LOCAL ] CHECK OPTION`

Cette option contrôle le comportement des vues actualisables automatiquement. Lorsque cette option est spécifiée, les commandes `INSERT` et `UPDATE` de la vue sont vérifiées pour s'assurer que les nouvelles lignes répondent à la condition définissant la vue (c'est-à-dire que les nouvelles lignes sont vérifiées pour s'assurer qu'elles sont visibles dans la vue). Dans le cas contraire, la mise à jour sera rejetée. Si l'option `CHECK OPTION` n'est pas spécifiée, les commandes `INSERT` et `UPDATE` de la vue sont autorisées à créer des lignes qui ne sont pas visibles dans la vue.

`LOCAL` : les nouvelles lignes ne sont vérifiées que par rapport aux conditions définies directement dans la vue elle-même. Les conditions définies sur les vues de base sous-jacentes ne sont pas vérifiées (sauf si elles spécifient également `CHECK OPTION`).

`CASCADED` : les nouvelles lignes sont vérifiées en fonction des conditions de la vue et de toutes les vues de base sous-jacentes. Si l'option `CHECK OPTION` est spécifiée et que ni `LOCAL` ni `CASCADED` n'est spécifié, alors `CASCADED` est supposé.

> **Note**
>
> L'option `CHECK OPTION` ne peut pas être utilisée avec les vues `RECURSIVE`. L'option `CHECK OPTION` est uniquement pris en charge que sur les vues qui peuvent être mises à jour automatiquement.

Remarques

Utilisez l'instruction `DROP VIEW` pour supprimer des vues.

Les noms et les types de données des colonnes de la vue doivent être soigneusement étudiés. Par exemple, CREATE VIEW vista AS SELECT « Hello World; » n'est pas recommandé, car le nom de colonne par défaut est `?column?;`. De plus, le type de données de colonne par défaut est `text`, ce qui n'est peut-être pas ce que vous vouliez.

Une meilleure approche consiste à spécifier explicitement le nom de la colonne et le type de données, tels que : `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

Par défaut, l'accès aux relations de base sous-jacentes référencées dans la vue est déterminé par les autorisations du propriétaire de la vue. Dans certains cas, cela peut être utilisé pour fournir un accès sécurisé, mais restreint aux tables sous-jacentes. Cependant, toutes les vues ne sont pas protégées contre la falsification.

- Si la propriété `security_invoker` de la vue est définie sur true, l'accès aux relations de base sous-jacentes est déterminé par les autorisations de l'utilisateur qui exécute la requête plutôt que par celles de la vue. Ainsi, l'utilisateur d'une vue d'invocation de sécurité doit disposer des autorisations appropriées sur la vue et ses relations de base sous-jacentes.
- Si l'une des relations de base sous-jacentes est une vue d'invocation de sécurité, elle sera traitée comme si elle avait été accessible directement depuis la requête d'origine. Ainsi, une vue d'invocation de sécurité vérifiera toujours ses relations de base sous-jacentes à l'aide des autorisations de l'utilisateur actuel, même si elle est accessible depuis une vue dépourvue de la propriété `security_invoker`.
- Les fonctions appelées dans la vue sont traitées de la même manière que si elles avaient été appelées directement depuis la requête utilisant la vue. Par conséquent, l'utilisateur d'une vue doit être autorisé à appeler toutes les fonctions utilisées par la vue. Les fonctions de la vue sont exécutées avec les privilèges de l'utilisateur exécutant la requête ou du propriétaire de la fonction, selon que les fonctions sont définies comme `SECURITY INVOKER` ou `SECURITY DEFINER`.

- L'utilisateur qui crée ou remplace une vue doit disposer de privilèges `USAGE` sur tous les schémas auxquels il est fait référence dans la requête de la vue, afin de pouvoir rechercher les objets référencés dans ces schémas.
- Lorsque `CREATE OR REPLACE VIEW` est utilisé sur une vue existante, seule la règle `SELECT` de la vue, ainsi que les paramètres `WITH ( ... )` et son option `CHECK OPTION` sont modifiés. Les autres propriétés de la vue, notamment la propriété, les autorisations et les règles autres que SELECT, restent inchangées. Vous devez être propriétaire de la vue pour la remplacer (cela inclut le fait d'être membre du rôle propriétaire).

Vues qui peuvent être mises à jour

Les vues simples peuvent être mises à jour automatiquement : le système autorisera l'utilisation des instructions `INSERT`, `UPDATE` et `DELETE` sur la vue de la même manière que sur une table normale. Une vue peut être automatiquement mise à jour si elle répond à toutes les conditions suivantes :

- La vue doit comporter exactement une entrée dans sa liste `FROM`, qui doit être une table ou une autre vue pouvant être mise à jour.
- La définition de la vue ne doit pas contenir de clauses `WITH`, `DISTINCT`, `GROUP BY`, `HAVING`, `LIMIT` ni `OFFSET` au niveau supérieur.
- La définition de la vue ne doit pas contenir d'opération définies (`UNION`, `INTERSECT` ni `EXCEPT`) au niveau supérieur.
- La liste de sélection de la vue ne doit pas contenir d'agrégats, de fonctions de fenêtrage ni de fonctions à renvoi d'ensemble.

Une vue pouvant être automatiquement mise à jour peut contenir un mélange de colonnes pouvant et ne pouvant pas être mises à jour. Une colonne peut être mise à jour s'il s'agit d'une simple référence à une colonne pouvant être mise à jour de la relation de base sous-jacente. Dans le cas contraire, la colonne est en lecture seule et une erreur se produit si une instruction `INSERT` ou `UPDATE` tente de lui attribuer une valeur.

Une vue plus complexe qui ne répond pas à toutes ces conditions est en lecture seule par défaut : le système n'autorise pas l'insertion, la mise à jour ni la suppression de la vue.

> **Note**
>
> L'utilisateur qui effectue l'insertion, la mise à jour ou la suppression de la vue doit disposer du privilège d'insertion, de mise à jour ou de suppression de la vue correspondant. Par défaut,

le propriétaire de la vue doit avoir les privilèges nécessaires sur les relations de base sous-jacentes, tandis que l'utilisateur qui effectue la mise à jour n'a besoin d'aucune autorisation sur les relations de base sous-jacentes. Toutefois, si security_invoker est défini sur true pour la vue, c'est l'utilisateur qui effectue la mise à jour, plutôt que le propriétaire de la vue, qui doit disposer des privilèges appropriés sur les relations de base sous-jacentes.

Exemples

Création d'une vue comprenant tous les films comiques.

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

Créer une vue avec `LOCAL CHECK OPTION`.

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

Créez une vue récursive.

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

Compatibilité

`CREATE OR REPLACE VIEW` est une extension du langage PostgreSQL. La clause `WITH ( ... )` est également une extension, tout comme les vues des barrières de sécurité et les vues d'invocation de sécurité. Aurora DSQL prend en charge ces extensions de langage.

`ALTER VIEW`

L'instruction `ALTER VIEW` permet de modifier diverses propriétés d'une vue existante, et Aurora DSQL prend en charge l'ensemble de la syntaxe PostgreSQL pour cette commande.

Syntaxe prise en charge

```
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`modifie diverses propriétés auxiliaires d'une vue. (Si vous souhaitez modifier la requête de définition de la vue, utilisez`CREATE OR REPLACE VIEW`.) Vous devez être propriétaire de la vue pour pouvoir l'utiliser`ALTER VIEW`. Pour modifier le schéma d'une vue, vous devez également disposer du privilège `CREATE` sur le nouveau schéma. Pour modifier le propriétaire, vous devez être en mesure de `SET ROLE` sur le nouveau rôle propriétaire, et ce rôle doit disposer du privilège `CREATE` sur le schéma de la vue.

Parameters

`name`

Nom (éventuellement qualifié selon le schéma) d'une vue existante.

`column_name`

Nom d'une colonne existante ou nouveau nom pour une colonne existante.

`IF EXISTS`

Ne génère pas d'erreur si la vue n'existe pas. Un avis est émis dans ce cas.

`SET/DROP DEFAULT`

Ces formulaires définissent ou suppriment la valeur par défaut d'une colonne. La valeur par défaut d'une colonne de vue est remplacée dans toute commande `INSERT` ou `UPDATE` dont la cible est la vue.

`new_owner`

Nom d'utilisateur du nouveau propriétaire de la vue.

`new_name`

Nouveau nom de la vue.

`new_schema`

Nouveau schéma de la vue.

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

Définit une option d'affichage. Les options suivantes sont prises en charge :

- `check_option (enum)` : modifie l'option de vérification de la vue. La valeur doit être `local` ou `cascaded`.
- `security_barrier (boolean)` : modifie la propriété de barrière de sécurité de la vue.
- `security_invoker (boolean)`- Modifie la propriété security-invoker de la vue.

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

Rétablit la valeur par défaut d'une option d'affichage.

Exemples

Renommer la vue `foo` en `bar` :

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

Attacher une valeur de colonne par défaut à une vue modifiable :

```
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
```

Compatibilité

`ALTER VIEW` est une extension PostgreSQL de la norme SQL prise en charge par Aurora DSQL.

DROP VIEW

L'instruction `DROP VIEW` supprime une vue existante. Aurora DSQL prend en charge la syntaxe PostgreSQL complète pour cette commande.

Syntaxe prise en charge

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

Description

`DROP VIEW` supprime une vue existante. Pour exécuter cette commande, vous devez être propriétaire de la vue.

Parameters

`IF EXISTS`

Ne génère pas d'erreur si la vue n'existe pas. Un avis est émis dans ce cas.

`name`

Nom (éventuellement qualifié selon le schéma) d'une vue à supprimer.

`CASCADE`

Supprime automatiquement les objets qui dépendent de la vue (comme d'autres vues), puis tous les objets qui dépendent de ces objets.

`RESTRICT`

Refuser de supprimer la vue si des objets en dépendent. Il s'agit de l'option par défaut.

Exemples

```
DROP VIEW kinds;
```

Compatibilité

Cette commande est conforme à la norme SQL, sauf que la norme n'autorise la suppression que d'une seule vue par commande, et à part l'option `IF EXISTS`, qui est une extension PostgreSQL prise en charge par Aurora DSQL.

Migration de PostgreSQL vers Aurora DSQL

Aurora DSQL est conçu pour être compatible avec PostgreSQL et prend en charge les fonctionnalités relationnelles de base telles que les transactions ACID, les index secondaires, les jointures et les

opérations DML standard. La plupart des applications PostgreSQL existantes peuvent migrer vers Aurora DSQL avec un minimum de modifications.

Cette section fournit des conseils pratiques pour la migration de votre application vers Aurora DSQL, notamment la compatibilité du framework, les modèles de migration et les considérations architecturales.

Compatibilité avec le framework et l'ORM

Aurora DSQL utilise le protocole filaire standard de PostgreSQL, garantissant ainsi la compatibilité avec les pilotes et les frameworks PostgreSQL. Les plus courants ORMs fonctionnent avec Aurora DSQL avec peu ou pas de modifications. Voir the section called "Adaptateurs Aurora DSQL" pour les implémentations de référence et les intégrations ORM disponibles.

Schémas de migration courants

Lors de la migration de PostgreSQL vers Aurora DSQL, certaines fonctionnalités fonctionnent différemment ou ont une syntaxe alternative. Cette section fournit des conseils sur les scénarios de migration courants.

Alternatives de fonctionnement du DDL

Aurora DSQL propose des alternatives modernes aux opérations DDL PostgreSQL traditionnelles :

Création d'index

À utiliser `CREATE INDEX ASYNC` plutôt que `CREATE INDEX` pour la création d'index non bloquants.

Avantage : création d'index sans interruption sur de grandes tables.

Suppression des données

Utiliser `DELETE FROM table_name` au lieu de`TRUNCATE`.

Alternative : Pour une récréation complète de la table, utilisez `DROP TABLE` puis`CREATE TABLE`.

Configuration du système

Aurora DSQL étant entièrement géré, la configuration est gérée automatiquement en fonction des modèles de charge de travail. Utilisez la console AWS de gestion ou l'API pour gérer les paramètres du cluster.

Avantage : pas besoin de réglage de la base de données ou de gestion des paramètres.

Modèles de conception de schémas

Adaptez ces modèles PostgreSQL courants pour assurer la compatibilité avec Aurora DSQL :

Modèles d'intégrité référentiels

Aurora DSQL prend en charge les relations et les `JOIN` opérations entre les tables. Pour garantir l'intégrité référentielle, implémentez la validation dans votre couche d'application. Cette conception s'aligne sur les modèles de bases de données distribuées modernes dans lesquels la validation de la couche application apporte plus de flexibilité et évite les problèmes de performance liés aux opérations en cascade.

Modèle : implémentez des contrôles d'intégrité référentielle dans votre couche d'application en utilisant des conventions de dénomination, une logique de validation et des limites de transaction cohérentes. De nombreuses applications à grande échelle préfèrent cette approche pour un meilleur contrôle de la gestion des erreurs et des performances.

Traitement temporaire des données

Utilisez CTEs des sous-requêtes ou des tables ordinaires avec une logique de nettoyage au lieu de tables temporaires.

Alternative : créez des tables avec des noms spécifiques à la session et nettoyez-les dans votre application.

Comprendre les différences architecturales

L'architecture distribuée et sans serveur d'Aurora DSQL se distingue intentionnellement de PostgreSQL traditionnel dans plusieurs domaines. Ces différences permettent d'exploiter les principaux avantages d'Aurora DSQL en termes de simplicité et d'évolutivité.

Modèle de base de données simplifié

Base de données unique par cluster

Aurora DSQL fournit une base de données intégrée nommée `postgres` par cluster.

Conseil de migration : si votre application utilise plusieurs bases de données, créez des clusters Aurora DSQL distincts pour une séparation logique ou utilisez des schémas au sein d'un seul cluster.

Pas de tables temporaires

Pour le traitement des données temporaires, vous DEVEZ utiliser des expressions de table (CTEs) et des sous-requêtes communes, qui fournissent des alternatives flexibles pour les requêtes complexes.

Alternative : à utiliser CTEs avec des `WITH` clauses pour des ensembles de résultats temporaires ou avec des tables ordinaires avec un nom unique pour les données spécifiques à la session.

Gestion automatique du stockage

Aurora DSQL élimine les tablespaces et la gestion manuelle du stockage. Le stockage s'adapte et s'optimise automatiquement en fonction de vos modèles de données.

Avantage : il n'est pas nécessaire de surveiller l'espace disque, de planifier l'allocation de stockage ou de gérer les configurations des tablespaces.

Modèles d'application modernes

Aurora DSQL encourage les modèles de développement d'applications modernes qui améliorent la maintenabilité et les performances :

Logique au niveau de l'application plutôt que des déclencheurs de base de données

Pour une fonctionnalité similaire à un déclencheur, implémentez une logique pilotée par les événements dans votre couche d'application.

Stratégie de migration : déplacez la logique de déclenchement vers le code de l'application, utilisez des architectures axées sur les événements avec des AWS services tels que EventBridge, ou implémentez des pistes d'audit à l'aide de la journalisation des applications.

Fonctions SQL pour le traitement des données

Aurora DSQL prend en charge les fonctions basées sur SQL, mais pas les langages procéduraux tels que PL/pgSQL.

Alternative : utilisez des fonctions SQL pour les transformations de données ou déplacez une logique complexe vers votre couche d'application ou vos fonctions AWS Lambda.

Un contrôle de simultanéité optimiste au lieu d'un verrouillage pessimiste

Aurora DSQL utilise un contrôle de simultanéité optimiste (OCC), une approche sans verrou qui diffère des mécanismes de verrouillage de base de données traditionnels. Au lieu d'acquérir des verrous qui bloquent d'autres transactions, Aurora DSQL permet aux transactions de se

poursuivre sans blocage et détecte les conflits au moment de la validation. Cela élimine les blocages et empêche les transactions lentes de bloquer d'autres opérations.

Principale différence : en cas de conflit, Aurora DSQL renvoie une erreur de sérialisation plutôt que d'obliger les transactions à attendre le verrouillage. Cela nécessite que les applications mettent en œuvre une logique de nouvelle tentative, similaire à la gestion des délais de verrouillage dans les bases de données traditionnelles, mais les conflits sont résolus immédiatement au lieu de provoquer des blocages d'attente.

Modèle de conception : implémentez une logique de transaction idempotente avec des mécanismes de nouvelle tentative. Concevez des schémas pour minimiser les conflits en utilisant des clés primaires aléatoires et en répartissant les mises à jour sur l'ensemble de votre plage de clés. Pour en savoir plus, consultez Contrôle de simultanéité dans Aurora DSQL.

Relations et intégrité référentielle

Aurora DSQL prend en charge les relations de clé étrangère entre les tables, y compris les `JOIN` opérations. Pour garantir l'intégrité référentielle, implémentez la validation dans votre couche d'application. Bien que le renforcement de l'intégrité référentielle puisse être utile, les opérations en cascade (comme les suppressions en cascade) peuvent créer des problèmes de performances inattendus. Par exemple, la suppression d'une commande comportant 1 000 articles devient une transaction de 1 001 lignes. C'est pourquoi de nombreux clients évitent les contraintes liées aux clés étrangères.

Modèle de conception : implémentez des contrôles d'intégrité référentiels dans votre couche d'application, utilisez d'éventuels modèles de cohérence ou tirez parti des AWS services pour la validation des données.

Simplifications opérationnelles

Aurora DSQL élimine de nombreuses tâches de maintenance de base de données traditionnelles, réduisant ainsi les frais d'exploitation :

Aucune maintenance manuelle requise

Aurora DSQL gère automatiquement l'optimisation du stockage, la collecte de statistiques et le réglage des performances. Les commandes de maintenance traditionnelles telles que celles-ci `VACUUM` sont gérées par le système.

Avantage : élimine le besoin de fenêtres de maintenance des bases de données, de planification du vide et de réglage des paramètres système.

Partitionnement et mise à l'échelle automatiques

Aurora DSQL partitionne et distribue automatiquement vos données en fonction des modèles d'accès. Généré par l' UUIDs utilisateur ou généré par l'application IDs pour une distribution optimale.

Conseil de migration : supprimez la logique de partitionnement manuel et laissez Aurora DSQL gérer la distribution des données. Généré par l' UUIDs utilisateur ou généré par l'application IDs pour une distribution optimale. Si votre application nécessite des identifiants séquentiels, consultez. Séquences et colonnes d'identité

Migration agentic avec des outils d'IA

Les agents de codage AI peuvent accélérer votre migration vers Aurora DSQL en analysant les schémas, en transformant le code et en exécutant des migrations DDL avec des contrôles de sécurité intégrés.

Utilisation de Kiro pour la migration

Les agents de codage tels que Kiro peuvent vous aider à analyser et à migrer votre code PostgreSQL vers Aurora DSQL :

- Analyse du schéma : téléchargez vos fichiers de schéma existants et demandez à Kiro d'identifier les problèmes de compatibilité potentiels et de suggérer des alternatives
- Transformation du code : fournissez le code de votre application et demandez à Kiro de vous aider à refactoriser la logique de déclenchement, à remplacer des séquences par des séquences ou à modifier UUIDs les modèles de transaction
- Planification de la migration : demandez à Kiro de créer un plan de step-by-step migration basé sur l'architecture spécifique de votre application
- Migrations DDL : exécutez des modifications de schéma à l'aide du modèle de recréation de table avec contrôles de sécurité intégrés et vérification utilisateur

Exemples d'instructions :

```
"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"
```

Migration DDL avec recréation de tables

Lorsque vous utilisez des agents d'intelligence artificielle avec le serveur Aurora DSQL MCP, certaines opérations ALTER TABLE utilisent un modèle de recréation de table qui migre vos données en toute sécurité. L'agent gère la complexité tout en vous tenant informé à chaque étape.

Les opérations suivantes utilisent le modèle de recréation des tables :

Opération	Approche
`DROP COLUMN`	Exclure la colonne du nouveau tableau
`ALTER COLUMN TYPE`	Type de données Cast pendant la migration
`ALTER COLUMN SET/DROP NOT NULL`	Contrainte de modification dans la nouvelle définition de table
`ALTER COLUMN SET/DROP DEFAULT`	Définir la valeur par défaut dans la nouvelle définition de table
`ADD/DROP CONSTRAINT`	Inclure ou supprimer une contrainte dans le nouveau tableau
`MODIFY PRIMARY KEY`	Définissez un nouveau PK avec validation de l'unicité
Diviser/fusionner des colonnes	Utilisez SPLIT_PART, SUBSTRING ou CONCAT

Les opérations ALTER TABLE suivantes sont prises en charge directement sans recréation de table :

- `ALTER TABLE ... RENAME COLUMN`— Renommer une colonne
- `ALTER TABLE ... RENAME TO`— Renommer une table
- `ALTER TABLE ... ADD COLUMN`— Ajoute une nouvelle colonne

Caractéristiques de sécurité : lors de l'exécution de migrations DDL, les agents d'intelligence artificielle présentent le plan de migration, vérifient la compatibilité des données, confirment le nombre de lignes et demandent une approbation explicite avant toute opération destructrice telle que DROP TABLE.

Migrations par lots : pour les tables de plus de 3 000 lignes, l'agent répartit automatiquement la migration par lots de 500 à 1 000 lignes afin de respecter les limites de transaction.

Serveur Aurora SQL MCP

Le serveur Aurora DSQL Model Context Protocol (MCP) permet aux assistants IA de se connecter directement à votre cluster Aurora DSQL et de rechercher la documentation Aurora DSQL. Cela permet à l'IA de :

- Analysez votre schéma existant et suggérez des modifications de migration
- Exécutez des migrations DDL avec le modèle de recréation de tables
- Tester les requêtes et vérifier la compatibilité lors de la migration
- Fournissez des up-to-date conseils précis basés sur la dernière documentation d'Aurora DSQL

Pour utiliser le serveur Aurora DSQL MCP avec des assistants AI, consultez les instructions de configuration du serveur Aurora DSQL MCP.

Considérations relatives à Aurora DSQL pour la compatibilité avec PostgreSQL

La prise en charge des fonctionnalités d'Aurora DSQL présente des différences par rapport à PostgreSQL autogéré, qui permettent son architecture distribuée, son fonctionnement sans serveur et son dimensionnement automatique. La plupart des applications fonctionnent dans les limites de ces différences sans modification.

Pour des considérations générales, consultez Considérations relatives à l'utilisation d Amazon Aurora DSQL. Pour les quotas et les limites, consultez Quotas de cluster et limites de base de données dans Amazon Aurora DSQL.

- Aurora DSQL utilise une seule base de données intégrée nommée `postgres` par cluster. Pour une séparation logique, créez des clusters Aurora DSQL distincts ou utilisez des schémas au sein d'un seul cluster.
- La `postgres` base de données utilise le codage de caractères UTF-8, qui fournit une large prise en charge des caractères internationaux.

- La base de données utilise uniquement le classement `C`.
- Aurora DSQL utilise `UTC` comme fuseau horaire du système. Postgres stocke toutes les dates et heures tenant compte du fuseau horaire en interne en UTC. Vous pouvez définir le paramètre de `TimeZone` configuration pour convertir la façon dont il est affiché pour le client et servir de valeur par défaut pour les entrées du client que le serveur utilisera pour convertir en UTC en interne.
- Le niveau d'isolement des transactions est défini dans PostgreSQL `Repeatable Read`.
- Les transactions sont soumises aux contraintes suivantes :
  - Les opérations DDL et DML nécessitent des transactions distinctes
  - Une transaction ne peut inclure qu'une seule instruction DDL
  - Une transaction peut modifier jusqu'à 3 000 lignes, quel que soit le nombre d'index secondaires
  - La limite de 3 000 lignes s'applique à toutes les instructions DML (`INSERT`, `UPDATE`, `DELETE`)
- Les connexions à la base de données expirent au bout d'une heure.
- Aurora DSQL gère les autorisations par le biais d'autorisations au niveau du schéma. Les utilisateurs administrateurs créent des schémas à l'aide de `CREATE SCHEMA` et accordent l'accès à l'aide `GRANT USAGE ON SCHEMA` de. Les utilisateurs administrateurs gèrent les objets dans le schéma public, tandis que les utilisateurs non administrateurs créent des objets dans des schémas créés par les utilisateurs pour définir clairement les limites de propriété. Pour de plus amples informations, veuillez consulter Autoriser les rôles de la base de données à utiliser SQL dans votre base de données.

Vous avez besoin d'aide pour la migration ?

Si vous rencontrez des fonctionnalités essentielles pour votre migration mais qui ne sont pas actuellement prises en charge dans Aurora DSQL, consultez Fournir des commentaires sur Amazon Aurora DSQL pour savoir comment partager des commentaires avec AWS.

Contrôle de simultanéité dans Aurora DSQL

La simultanéité permet à plusieurs sessions d'accéder aux données et de les modifier simultanément sans compromettre l'intégrité et la cohérence des données. Aurora DSQL assure la compatibilité avec PostgreSQL tout en mettant en œuvre un mécanisme de contrôle de simultanéité moderne et sans verrou. Aurora DSQL assure une conformité totale à ACID grâce à l'isolement des instantanés, garantissant ainsi la cohérence et la fiabilité des données.

L'un des principaux avantages d'Aurora DSQL est son architecture sans verrou, qui élimine les problèmes courants liés aux performances des bases de données. Aurora DSQL empêche les transactions lentes de bloquer d'autres opérations et élimine le risque de blocages. Cette approche rend Aurora DSQL particulièrement utile pour les applications à haut débit où les performances et la capacité de mise à l'échelle sont essentielles.

Conflits de transaction

Aurora DSQL utilise un contrôle de simultanéité optimiste (OCC), qui fonctionne différemment des systèmes traditionnels basés sur des verrous. Au lieu d'utiliser des verrous, l'OCC évalue les conflits au moment de la validation. Lorsque plusieurs transactions entrent en conflit lors de la mise à jour de la même ligne, Aurora DSQL gère les transactions comme suit :

- La transaction dont l'heure de validation est la plus proche est traitée par Aurora DSQL.
- Les transactions en conflit reçoivent une erreur de sérialisation PostgreSQL, indiquant la nécessité d'une nouvelle tentative.

Concevez vos applications de manière à mettre en œuvre une logique de nouvelle tentative pour gérer les conflits. Le modèle de conception idéal est idempotent, permettant une nouvelle tentative de transaction comme premier recours dans la mesure du possible. La logique recommandée est similaire à la logique d'abandon et de nouvelle tentative dans une situation de blocage ou de temporisation standard de PostgreSQL. Cependant, l'OCC exige que vos applications appliquent cette logique plus fréquemment.

Directives pour optimiser les performances des transactions

Pour optimiser les performances, réduisez les risques de conflits sur des clés uniques ou sur de petites plages de clés. Pour atteindre cet objectif, concevez votre schéma de manière à répartir les mises à jour sur l'ensemble de votre plage de clés de cluster en suivant les directives suivantes :

- Choisissez une clé primaire aléatoire pour vos tables.
- Évitez les modèles qui accroissent le risque de conflit sur les clés individuelles. Cette approche garantit des performances optimales même lorsque le volume des transactions augmente.

Transactions DDL et distribuées dans Aurora DSQL

Le langage de définition des données (DDL) se comporte différemment dans Aurora DSQL et dans PostgreSQL. Aurora DSQL propose une couche de base de données distribuée et sans partage Multi-AZ, construite sur des flottes de calcul et de stockage à plusieurs locataires. Comme il n'existe pas de nœud de base de données primaire ni de leader unique, le catalogue de base de données est distribué. Ainsi, Aurora DSQL gère les modifications du schéma DDL sous forme de transactions distribuées.

Plus précisément, DDL se comporte différemment dans Aurora DSQL comme suit :

Erreurs de contrôle de simultanéité

Aurora DSQL renvoie une erreur de violation du contrôle de simultanéité si vous exécutez une transaction alors qu'une autre met à jour une ressource. Par exemple, considérez la séquence d'actions suivante :

1. Au cours de la session 1, un utilisateur ajoute une colonne à la table `mytable`.
2. Au cours de la session 2, un utilisateur tente d'insérer une ligne dans `mytable`.

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

DDL et DML dans la même transaction

Les transactions dans Aurora DSQL ne peuvent contenir qu'une seule instruction DDL et ne peuvent pas contenir à la fois des instructions DDL et DML. Cette restriction signifie que vous ne pouvez pas créer de table et insérer des données dans la même table au cours de la même transaction. Par exemple, Aurora DSQL prend en charge les transactions séquentielles suivantes.

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

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

Aurora DSQL ne prend pas en charge la transaction suivante, qui inclut à la fois les instructions `CREATE` et `INSERT`.

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

DDL asynchrone

Dans PostgreSQL standard, les opérations DDL telles que `CREATE INDEX` verrouille la table concernée, la rendant indisponible pour les lectures et les écritures provenant d'autres sessions. Dans Aurora DSQL, ces instructions DDL s'exécutent de manière asynchrone à l'aide d'un gestionnaire d'arrière-plan. L'accès à la table concernée n'est pas bloqué. Ainsi, DDL sur de grandes tables peut fonctionner sans durée d'indisponibilité ni impact sur les performances. Pour plus d'informations sur l'utilisation du gestionnaire de tâches asynchrones dans Aurora DSQL, consultez Index asynchrones dans Aurora DSQL.

Clés primaires dans Aurora DSQL

Dans Aurora DSQL, une clé primaire est une fonctionnalité qui organise physiquement les données d'une table. C'est similaire à l'opération `CLUSTER` dans PostgreSQL ou à un index en cluster dans d'autres bases de données. Lorsque vous définissez une clé primaire, Aurora DSQL crée un index qui inclut toutes les colonnes de la table. La structure de clé primaire d'Aurora DSQL garantit un accès et une gestion efficaces des données.

Structure et stockage des données

Lorsque vous définissez une clé primaire, Aurora DSQL stocke les données des tables dans l'ordre des clés primaires. Cette structure organisée par index permet une recherche par clé primaire pour récupérer directement toutes les valeurs des colonnes, au lieu de suivre un pointeur vers les données comme dans un index d'arbre B traditionnel. Contrairement à l'opération `CLUSTER` de PostgreSQL, qui ne réorganise les données qu'une seule fois, Aurora DSQL maintient cet ordre automatiquement et en continu. Cette approche permet d'améliorer les performances des requêtes qui reposent sur l'accès par clé primaire.

Aurora DSQL utilise également la clé primaire pour générer une clé unique à l'échelle du cluster pour chaque ligne des tables et des index. Cette clé unique est également à la base de la gestion des données distribuées. Elle permet le partitionnement automatique des données sur plusieurs nœuds, prenant en charge un stockage évolutif et une simultanéité élevée. Par conséquent, la structure

de clé primaire permet à Aurora DSQL de s'adapter automatiquement et de gérer efficacement les charges de travail simultanées.

Directives pour choisir une clé primaire

Lorsque vous choisissez et utilisez une clé primaire dans Aurora DSQL, tenez compte des directives suivantes :

- Définissez une clé primaire lorsque vous créez une table. Vous ne pouvez pas modifier cette clé ni ajouter une nouvelle clé primaire ultérieurement. La clé primaire fait partie de la clé à l'échelle du cluster utilisée pour le partitionnement des données et la mise à l'échelle automatique du débit d'écriture. Si vous ne spécifiez pas de clé primaire, Aurora DSQL attribuera un ID fictif masqué.
- Pour les tables présentant des volumes d'écriture élevés, évitez d'utiliser des nombres entiers qui augmentent de façon monotone comme clés primaires. Cela peut entraîner des problèmes de performances en dirigeant tous les nouveaux inserts vers une seule partition. Utilisez plutôt des clés primaires à distribution aléatoire pour garantir une répartition uniforme des écritures sur les partitions de stockage.
- Pour les tables qui changent rarement ou qui sont en lecture seule, vous pouvez utiliser une clé ascendante. Les horodatages ou les numéros de séquence sont des exemples de clés ascendantes. Une clé dense comporte de nombreuses valeurs étroitement espacées ou dupliquées. Vous pouvez utiliser une clé ascendante même si elle est dense, car les performances d'écriture sont moins critiques.
- Si une analyse complète du tableau ne répond pas à vos exigences de performance, choisissez une méthode d'accès plus efficace. Dans la plupart des cas, cela implique d'utiliser une clé primaire qui correspond à votre clé de jointure et de recherche la plus courante dans les requêtes.
- La taille maximale combinée des colonnes utilisées dans une clé primaire est de 1 kibioctet. Pour plus d'informations, consultez Limites de base de données dans Aurora DSQL et Types de données pris en charge dans Aurora DSQL.
- Vous pouvez inclure jusqu'à 8 colonnes dans une clé primaire ou un index secondaire. Pour plus d'informations, consultez Limites de base de données dans Aurora DSQL et Types de données pris en charge dans Aurora DSQL.

Séquences et colonnes d'identité

Les séquences et les colonnes d'identité génèrent des valeurs entières et sont utiles lorsque des identifiants compacts ou lisibles par l'homme sont nécessaires. Ces valeurs impliquent le comportement d'allocation et de mise en cache décrit dans la `CREATE SEQUENCE` documentation.

Rubriques

- Fonctions de manipulation de séquence
- Colonnes d'identité
- Utilisation de séquences et de colonnes d'identité

Fonctions de manipulation de séquence

Cette section décrit les fonctions permettant d'opérer sur des objets de séquence, également appelés générateurs de séquences ou simplement séquences. Les objets de séquence sont des tables spéciales à une seule ligne créées avec`CREATE SEQUENCE`. Les objets de séquence sont couramment utilisés pour générer des identifiants uniques pour les lignes d'une table. Les fonctions de séquence fournissent des méthodes simples et sécurisées pour plusieurs utilisateurs pour obtenir des valeurs de séquence successives à partir d'objets de séquence.

> **Important**
>
> Lorsque vous utilisez des séquences, la valeur du cache doit être soigneusement prise en compte. Pour plus d'informations, consultez la légende Important sur la `CREATE SEQUENCE` page.
> Pour obtenir des conseils sur la meilleure façon d'utiliser les séquences basées sur les modèles de charge de travail, voirUtilisation de séquences et de colonnes d'identité.

Fonction	Description
`nextval ( regclass ) # bigint`	Avance l'objet de séquence jusqu'à sa valeur suivante et renvoie cette valeur. Cela se fait de manière atomique : même si plusieurs sessions s'exécutent `nextval` simultanément, chacune recevra en toute sécurité une valeur de séquence distincte. Si l'objet de séquence a

Fonction	Description
	été créé avec des paramètres par défaut, les `nextval` appels successifs renverront des valeurs croissantes commençant par 1. D'autres comportements peuvent être obtenus en utilisant les paramètres appropriés dans la `CREATE SEQUENCE` commande. Cette fonction requiert USAGE ou UPDATE privilégie la séquence.

`setval ( regclass, bigint [, boolean ] ) # bigint`	Définit la valeur actuelle de l'objet de séquence, et éventuellement son `is_called` indicateur. Le formulair e à deux paramètres définit le `last_value` champ de la séquence à la valeur spécifiée et définit son `is_called` champ à la valeur spécifié`true`, ce qui signifie que le suivant `nextval` fera avancer la séquence avant de renvoyer une valeur. La valeur qui sera signalée `currval` est également définie sur la valeur spécifiée. Dans le formulaire à trois paramètre s, il `is_called` peut être défini sur `true` ou`false`. `true`a le même effet que le formulaire à deux paramètre s. S'il est défini sur`false`, le suivant `nextval` renverra exactement la valeur spécifiée, et l'avancement de la séquence commence par ce qui suit`nextval`. De plus, la valeur indiquée par `currval` n'est pas modifiée ici. Par exemple :
	`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`
	Le résultat renvoyé par `setval` est simplement la valeur de son deuxième argument. Cette fonction nécessite un UPDATE privilège sur la séquence.

Fonction	Description
`currval ( regclass ) # bigint`	Renvoie la dernière valeur obtenue par `nextval` pour cette séquence dans la session en cours. (Une erreur est signalée si cette séquence n'`nextval`a jamais été appelée au cours de cette session.) Comme cela renvoie une valeur locale à la session, cela donne une réponse prévisible, que d'autres sessions aient été exécutées ou non `nextval` parce que la session en cours l'a fait. Cette fonction requiert `USAGE` ou `SELECT` privilégie la séquence.
`lastval () # bigint`	Renvoie la dernière valeur renvoyée par la transacti on `nextval` en cours. Cette fonction est identique à`currval`, sauf qu'au lieu de prendre le nom de la séquence comme argument, elle fait référence à la séquence `nextval` la plus récente appliquée dans la transaction en cours. L'appel est une erreur `lastval` s'il `nextval` n'a pas encore été appelé dans le cadre de la transaction en cours. Cette fonction requiert `USAGE` ou `SELECT` privilégie la dernière séquence utilisée.

Warning

La valeur obtenue par `nextval` n'est pas récupérée pour être réutilisée si la transaction d'appel est abandonnée ultérieurement. Cela signifie que les annulations de transactions ou les pannes de base de données peuvent entraîner des lacunes dans la séquence des valeurs attribuées. Cela peut également se produire sans interruption de transaction. Par exemple, une `ON CONFLICT` clause `INSERT` with an calculera le to-be-inserted tuple, y compris en effectuant les `nextval` appels requis, avant de détecter tout conflit qui l'obligerait à suivre la `ON CONFLICT` règle à la place. Ainsi, les objets de séquence d'Aurora DSQL ne peuvent pas être utilisés pour obtenir des séquences « sans interruption ».
De même, les modifications d'état de séquence effectuées par `setval` sont immédiatement visibles pour les autres transactions et ne sont pas annulées si la transaction appelante est annulée.

La séquence à exécuter par une fonction de séquence est spécifiée par un `regclass` argument, qui est simplement l'OID de la séquence dans le catalogue `pg_class` système. Cependant, il n'est pas nécessaire de rechercher l'OID manuellement, car le convertisseur d'entrée du type de `regclass` données fera le travail à votre place. Consultez la documentation de PostgreSQL sur les types d'identificateurs d'objets pour plus de détails.

Colonnes d'identité

> **Important**
>
> Lorsque vous utilisez des colonnes d'identité, la valeur du cache doit être soigneusement prise en compte. Pour plus d'informations, consultez la légende Important sur la `CREATE SEQUENCE` page.
> Pour obtenir des conseils sur la meilleure façon d'utiliser les colonnes d'identité en fonction des modèles de charge de travail, voirUtilisation de séquences et de colonnes d'identité.

Une colonne d'identité est une colonne spéciale générée automatiquement à partir d'une séquence implicite. Il peut être utilisé pour générer des valeurs clés. Pour créer une colonne d'identité, utilisez la `GENERATED ... AS IDENTITY` clause dans`CREATE TABLE`, par exemple :

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

ou bien :

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

Pour plus d'informations, consultez `CREATE TABLE`.

Si une `INSERT` commande est exécutée sur la table contenant la colonne d'identité et qu'aucune valeur n'est explicitement spécifiée pour la colonne d'identité, une valeur générée par la séquence implicite est insérée. Par exemple, avec les définitions précédentes et en supposant que des colonnes supplémentaires soient appropriées, en écrivant :

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

générerait des valeurs pour la `id` colonne commençant à 1 et produirait les données de table suivantes :

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

Vous pouvez également spécifier le mot-clé `DEFAULT` à la place d'une valeur pour demander explicitement la valeur générée par la séquence :

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

De même, le mot-clé `DEFAULT` peut être utilisé dans `UPDATE` les commandes.

Ainsi, à bien des égards, une colonne d'identité se comporte comme une colonne avec une valeur par défaut.

Les clauses `ALWAYS` et la définition `BY DEFAULT` de colonne déterminent la manière dont les valeurs spécifiées explicitement par l'utilisateur sont traitées dans `UPDATE` les commandes `INSERT` et. Dans une `INSERT` commande, si elle `ALWAYS` est sélectionnée, une valeur spécifiée par l'utilisateur n'est acceptée que si l'`INSERT`instruction le précise`OVERRIDING SYSTEM VALUE`. Si cette option `BY DEFAULT` est sélectionnée, la valeur spécifiée par l'utilisateur est prioritaire. Ainsi, l'utilisation `BY DEFAULT` entraîne un comportement plus proche des valeurs par défaut, où la valeur par défaut peut être remplacée par une valeur explicite, tout en `ALWAYS` offrant une protection supplémentaire contre l'insertion accidentelle d'une valeur explicite.

Le type de données d'une colonne d'identité doit être l'un des types de données pris en charge par les séquences. (Consultez `CREATE SEQUENCE`.) Les propriétés de la séquence associée peuvent être spécifiées lors de la création d'une colonne d'identité (voir`CREATE TABLE`) ou modifiées ultérieurement (voir`ALTER TABLE`).

Une colonne d'identité est automatiquement marquée comme`NOT NULL`. Une colonne d'identité ne garantit toutefois pas l'unicité. (Une séquence renvoie normalement des valeurs uniques, mais une séquence peut être réinitialisée ou des valeurs peuvent être insérées manuellement dans la colonne

d'identité, comme indiqué précédemment.) L'unicité devrait être appliquée à l'aide d'une `UNIQUE` contrainte `PRIMARY KEY` or.

Utilisation de séquences et de colonnes d'identité

Cette section vous aide à comprendre comment utiliser au mieux les séquences et les colonnes d'identité en fonction des modèles de charge de travail.

> **Important**
>
> Consultez la légende Important de la `CREATE SEQUENCE` page pour plus de détails sur le comportement d'allocation et de mise en cache.

Choix des types d'identificateurs

Amazon Aurora DSQL prend en charge à la fois les identifiants basés sur l'UUID et les valeurs entières générées à l'aide de séquences ou de colonnes d'identité. Ces options diffèrent quant à la manière dont les valeurs sont allouées et à leur mise à l'échelle en fonction de la charge.

Les valeurs UUID peuvent être générées sans coordination et conviennent parfaitement aux charges de travail dans lesquelles des identifiants sont créés fréquemment ou au cours de nombreuses sessions. Amazon Aurora DSQL étant conçu pour un fonctionnement distribué, il est souvent avantageux d'éviter la coordination. C'est pourquoi UUIDs ils sont recommandés comme type d'identifiant par défaut, en particulier pour les clés primaires dans les charges de travail où l'évolutivité est importante et où un ordre strict des identifiants n'est pas requis.

Les séquences et les colonnes d'identité génèrent des valeurs entières compactes qui sont pratiques pour les identificateurs lisibles par l'homme, les rapports et les interfaces externes. Lorsque les identifiants numériques sont préférés pour des raisons d'utilisabilité ou d'intégration, envisagez d'utiliser une séquence ou une colonne d'identité en combinaison avec des identifiants basés sur l'UUID. Lorsque des séquences entières ou des valeurs d'identité sont requises, le choix d'une taille de cache appropriée devient un élément important de la conception de la charge de travail. Consultez la section suivante pour savoir comment choisir une taille de cache.

Choix d'une taille de cache

La sélection d'une valeur de cache appropriée est un élément important pour utiliser efficacement les séquences et les colonnes d'identité. Le paramètre du cache détermine le comportement de

l'allocation des identifiants sous charge, influençant à la fois le débit du système et la mesure dans laquelle les valeurs reflètent l'ordre d'allocation.

Une taille de cache plus importante **`CACHE >= 65536`** est idéale lorsque :

- Les identifiants sont générés à haute fréquence
- De nombreuses sessions s'insèrent simultanément
- La charge de travail peut tolérer des écarts et des effets de commande visibles

Par exemple, les charges de travail d'ingestion d'événements à volume élevé (comme l'IoT ou la télémétrie), ainsi que les identifiants opérationnels tels que l'exécution des tâches IDs, les références de dossiers d'assistance ou les numéros de commande internes bénéficient généralement de tailles de cache plus importantes, où des identifiants sont générés fréquemment et un ordre strict n'est pas requis.

Une taille de cache de 1 est mieux alignée lorsque :

- Les taux d'allocation sont relativement bas
- Les identifiants devraient suivre de plus près l'ordre d'allocation au fil du temps
- Il est plus important de minimiser les écarts que de maximiser le débit

Les charges de travail telles que l'attribution de numéros de compte ou de référence, pour lesquelles les identifiants sont générés moins souvent et où un ordre plus serré est souhaitable, sont mieux adaptées à une taille de cache de 1.

Index asynchrones dans Aurora DSQL

La commande `CREATE INDEX ASYNC` crée un index sur une ou plusieurs colonnes d'une table spécifiée. Cette commande est une opération DDL asynchrone qui ne bloque pas les autres transactions. Lorsque vous exécutez `CREATE INDEX ASYNC`, Aurora DSQL renvoie immédiatement un `job_id`.

Vous pouvez surveiller l'état de cette tâche asynchrone à l'aide de la vue système `sys.jobs`. Pendant que le travail de création d'index est en cours, vous pouvez utiliser les procédures et commandes suivantes :

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

Bloque la session en cours jusqu'à ce que la tâche spécifiée soit terminée ou échoue. Renvoie une valeur booléenne indiquant la réussite ou l'échec.

`DROP INDEX`

Annule une tâche de création d'index en cours.

Lorsque la création de l'index asynchrone est terminée, Aurora DSQL met à jour le catalogue système pour marquer l'index comme actif.

> **Note**
>
> Notez que les transactions simultanées accédant à des objets dans le même espace de noms au cours de cette mise à jour peuvent rencontrer des erreurs de simultanéité.

Lorsqu'Aurora DSQL termine une tâche d'indexation asynchrone, il met à jour le catalogue système pour indiquer que l'index est actif. Si d'autres transactions font référence aux objets du même espace de noms à ce stade, une erreur de simultanéité peut apparaître.

Syntaxe

`CREATE INDEX ASYNC` utilise la syntaxe suivante.

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

Parameters

`UNIQUE`

Indique à Aurora DSQL de vérifier les valeurs dupliquées dans la table lors de la création de l'index et à chaque fois que vous ajoutez des données. Si vous spécifiez ce paramètre, les opérations d'insertion et de mise à jour susceptibles d'entraîner des doublons génèrent une erreur.

`IF NOT EXISTS`

Indique qu'Aurora DSQL ne doit pas générer d'exception si un index portant le même nom existe déjà. Dans ce cas, Aurora DSQL ne crée pas le nouvel index. Notez que l'index que vous essayez de créer peut avoir une structure très différente de celle de l'index existant. Si vous spécifiez ce paramètre, le nom de l'index est obligatoire.

`name`

Nom de l'index. Vous ne pouvez pas inclure le nom de votre schéma dans ce paramètre.

Aurora DSQL crée l'index dans le même schéma que sa table parent. Le nom de l'index doit être distinct du nom de tout autre objet, tel qu'une table ou un index, dans le schéma.

Si vous ne spécifiez pas de nom, Aurora DSQL le génère automatiquement en fonction du nom de la table parent et de la colonne indexée. Par exemple, si vous exécutez `CREATE INDEX ASYNC on table1 (col1, col2)`, Aurora DSQL nomme automatiquement l'index `table1_col1_col2_idx`.

`NULLS FIRST | LAST`

Ordre de tri des colonnes nulles et non nulles. `FIRST` indique qu'Aurora DSQL doit trier les colonnes nulles avant les colonnes non nulles. `LAST` indique qu'Aurora DSQL doit trier les colonnes nulles après les colonnes non nulles.

`INCLUDE`

Liste des colonnes à inclure dans l'index en tant que colonnes non clés. Vous ne pouvez pas utiliser une colonne non clé dans une qualification de recherche par analyse d'index. Aurora DSQL ignore la colonne en termes d'unicité pour un index.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Spécifie si Aurora DSQL doit considérer les valeurs nulles comme distinctes dans un index unique. La valeur par défaut est `DISTINCT`, ce qui signifie qu'un index unique peut contenir plusieurs valeurs nulles dans une colonne. `NOT DISTINCT` indique qu'un index ne peut pas contenir plusieurs valeurs nulles dans une colonne.

Notes d'utilisation

Considérez les directives suivantes :

- La commande `CREATE INDEX ASYNC` n'introduit pas de verrous. Cela n'affecte pas non plus la table de base qu'Aurora DSQL utilise pour créer l'index.
- Lors des opérations de migration de schéma, la procédure `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* est utile. Cela garantit que les opérations DDL et DML suivantes ciblent l'index nouvellement créé.
- Chaque fois qu'Aurora DSQL exécute une nouvelle tâche asynchrone, il vérifie la vue `sys.jobs` et supprime les tâches dont le statut est `completed` ou `failed` pendant plus de 30 minutes. Ainsi, `sys.jobs` affiche principalement les tâches en cours et ne contient aucune information sur les anciennes tâches.
- Si Aurora DSQL ne parvient pas à créer un index asynchrone, l'index reste `INVALID`. Pour les index uniques, les opérations DML sont soumises à des contraintes d'unicité jusqu'à ce que vous supprimiez l'index. Nous vous recommandons de supprimer les index non valides et de les recréer.

Création d'un index : exemple

L'exemple suivant montre comment créer un schéma, une table, puis un index.

1. Créez une nouvelle table nommée `test.departments`.

```
CREATE SCHEMA test;

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

2. Insérez une ligne dans la table.

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

3. Créez un index asynchrone.

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

   La commande `CREATE INDEX` renvoie un identifiant de tâche, comme indiqué ci-dessous.

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

`job_id` indique qu'Aurora DSQL a soumis une nouvelle tâche pour créer l'index. Vous pouvez utiliser la procédure `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* pour bloquer d'autres tâches au cours de la session jusqu'à cette tâche soit terminée ou expire.

Interrogation de l'état de création de l'index : exemple

Interrogez la vue système `sys.jobs` pour vérifier l'état de création de votre index, comme illustré dans l'exemple suivant.

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

Aurora DSQL renvoie une réponse semblable à ce qui suit.

```
           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
```

La colonne d'état peut avoir l'une des valeurs suivantes.

Statut	Description
`submitted`	La tâche est envoyée, mais Aurora DSQL n'a pas encore commencé à la traiter.
`processing`	Aurora DSQL est en train de traiter la tâche.
`failed`	La tâche a échoué. Consultez la colonne « détails » pour plus d'informations. Si Aurora DSQL ne parvient pas à créer l'index, Aurora DSQL ne supprime pas automatiquement la définition de l'index. Vous devez supprimer manuellement l'index à l'aide de la commande `DROP INDEX`.
`completed`	Aurora DSQL a terminé la tâche avec succès.

Vous pouvez également interroger l'état de l'index via les tables du catalogue `pg_index` et `pg_class`. Plus précisément, les attributs `indisvalid` et `indisimmediate` peuvent vous indiquer dans quel état se trouve votre index. Lors de la création de votre index par Aurora DSQL, celui-ci possède l'état initial `INVALID`. L'indicateur `indisvalid` de l'index renvoie `FALSE` ou `f`, ce qui indique que l'index n'est pas valide. Si l'indicateur revient à `TRUE` ou `t`, l'index est prêt.

```
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)
```

Défaillances de création d'index uniques

Si votre tâche de création d'index unique asynchrone indique un état d'échec avec le détail `Found duplicate key while validating index for UCVs`, cela indique qu'un index unique n'a pas pu être créé en raison de violations des contraintes d'unicité.

Résolution des problèmes de défaillances de création d'index uniques

1. Supprimez toutes les lignes de votre table principale contenant des entrées dupliquées pour les clés spécifiées dans votre index secondaire unique.
2. Supprimez l'index défaillant.
3. Émettez une nouvelle commande de création d'index.

Détection des violations d'unicité dans les tables principales

La requête SQL suivante vous aide à identifier les valeurs dupliquées dans une colonne spécifiée de votre table. Cela est particulièrement utile lorsque vous devez appliquer l'unicité à une colonne qui

n'est pas actuellement définie comme clé primaire ou qui n'a pas de contrainte unique, telle que les adresses e-mail dans une table utilisateur.

Les exemples ci-dessous montrent comment créer un exemple de table des utilisateurs, la remplir avec des données de test contenant des doublons connus, puis exécuter la requête de détection.

Définir le schéma de table

```
-- 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
);
```

Insérer des exemples de données comprenant des ensembles d'adresses e-mail dupliquées

```
-- 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');
```

Exécuter une requête de détection des doublons

```
-- 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;
```

Afficher tous les enregistrements contenant des adresses e-mail dupliquées

```
 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)
```

Si nous devions essayer l'instruction de création d'index maintenant, elle échouerait :

```
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)
```

Tables et commandes système dans Aurora DSQL

Consultez les sections suivantes pour en savoir plus sur les tables système et les catalogues pris en charge dans Aurora DSQL, ainsi que sur les requêtes utiles pour récupérer des informations sur le système, telles que la version.

Tables système

Aurora DSQL est compatible avec PostgreSQL, de sorte que de nombreuses tables du catalogue système et de nombreuses vues de PostgreSQL existent également dans Aurora DSQL.

Tables et vues du catalogue PostgreSQL importantes

Le tableau suivant décrit les tables et les vues les plus courantes que vous pouvez utiliser dans Aurora DSQL.

Nom	Description
`pg_namespace`	Informations sur tous les schémas
`pg_tables`	Informations sur toutes les tables
`pg_attribute`	Informations sur tous les attributs
`pg_views`	Informations sur les vues (pré)-définies
`pg_class`	Décrit toutes les tables, toutes les colonnes, tous les index et tous les objets similaires
`pg_stats`	Vue des statistiques du planificateur
`pg_user`	Informations sur les utilisateurs
`pg_roles`	Informations sur les utilisateurs et les groupes
`pg_indexes`	Répertorie tous les index
`pg_constraint`	Répertorie les contraintes sur les tables

Tables de catalogue prises en charge et non prises en charge

Le tableau suivant indique les tables prises en charge et celles qui ne le sont pas dans Aurora DSQL.

Nom	Applicable à Aurora DSQL
`pg_aggregate`	Non
`pg_am`	Oui
`pg_amop`	Non
`pg_amproc`	Non
`pg_attrdef`	Oui
`pg_attribute`	Oui
`pg_authid`	Non (utiliser `pg_roles`)
`pg_auth_members`	Oui
`pg_cast`	Oui
`pg_class`	Oui
`pg_collation`	Oui
`pg_constraint`	Oui
`pg_conversion`	Non
`pg_database`	Non
`pg_db_role_setting`	Oui
`pg_default_acl`	Oui
`pg_depend`	Oui
`pg_description`	Oui

Nom	Applicable à Aurora DSQL
pg_enum	Non
pg_event_trigger	Non
pg_extension	Non
pg_foreign_data_wrapper	Non
pg_foreign_server	Non
pg_foreign_table	Non
pg_index	Oui
pg_inherits	Oui
pg_init_privs	Non
pg_language	Non
pg_largeobject	Non
pg_largeobject_metadata	Oui
pg_namespace	Oui
pg_opclass	Non
pg_operator	Oui
pg_opfamily	Non
pg_parameter_acl	Oui
pg_partitioned_table	Non
pg_policy	Non
pg_proc	Non

Nom	Applicable à Aurora DSQL
pg_publication	Non
pg_publication_namespace	Non
pg_publication_rel	Non
pg_range	Oui
pg_replication_origin	Non
pg_rewrite	Non
pg_seclabel	Non
pg_sequence	Non
pg_shdepend	Oui
pg_shdescription	Oui
pg_shseclabel	Non
pg_statistic	Oui
pg_statistic_ext	Non
pg_statistic_ext_data	Non
pg_subscription	Non
pg_subscription_rel	Non
pg_tablespace	Non
pg_transform	Non
pg_trigger	Non
pg_ts_config	Oui

Nom	Applicable à Aurora DSQL
pg_ts_config_map	Oui
pg_ts_dict	Oui
pg_ts_parser	Oui
pg_ts_template	Oui
pg_type	Oui
pg_user_mapping	Non

Vues système prises en charge et non prises en charge

Le tableau suivant indique les vues prises en charge et celles qui ne le sont pas dans Aurora DSQL.

Nom	Applicable à Aurora DSQL
pg_available_extensions	Non
pg_available_extension_versions	Non
pg_backend_memory_contexts	Oui
pg_config	Non
pg_cursors	Non
pg_file_settings	Non
pg_group	Oui
pg_hba_file_rules	Non
pg_ident_file_mappings	Non
pg_indexes	Oui

Nom	Applicable à Aurora DSQL
pg_locks	Non
pg_matviews	Non
pg_policies	Non
pg_prepared_statements	Non
pg_prepared_xacts	Non
pg_publication_tables	Non
pg_replication_origin_status	Non
pg_replication_slots	Non
pg_roles	Oui
pg_rules	Non
pg_seclabels	Non
pg_sequences	Non
pg_settings	Oui
pg_shadow	Oui
pg_shmem_allocations	Oui
pg_stats	Oui
pg_stats_ext	Non
pg_stats_ext_exprs	Non
pg_tables	Oui
pg_timezone_abbrevs	Oui

Nom	Applicable à Aurora DSQL
pg_timezone_names	Oui
pg_user	Oui
pg_user_mappings	Non
pg_views	Oui
pg_stat_activity	Non
pg_stat_replication	Non
pg_stat_replication_slots	Non
pg_stat_wal_receiver	Non
pg_stat_recovery_prefetch	Non
pg_stat_subscription	Non
pg_stat_subscription_stats	Non
pg_stat_ssl	Oui
pg_stat_gssapi	Non
pg_stat_archiver	Non
pg_stat_io	Non
pg_stat_bgwriter	Non
pg_stat_wal	Non
pg_stat_database	Non
pg_stat_database_conflicts	Non
pg_stat_all_tables	Non

Nom	Applicable à Aurora DSQL
pg_stat_all_indexes	Non
pg_statio_all_tables	Non
pg_statio_all_indexes	Non
pg_statio_all_sequences	Non
pg_stat_slru	Non
pg_statio_user_tables	Non
pg_statio_user_sequences	Non
pg_stat_user_functions	Non
pg_stat_user_indexes	Non
pg_stat_progress_analyze	Non
pg_stat_progress_basebackup	Non
pg_stat_progress_cluster	Non
pg_stat_progress_create_index	Non
pg_stat_progress_vacuum	Non
pg_stat_sys_indexes	Non
pg_stat_sys_tables	Non
pg_stat_xact_all_tables	Non
pg_stat_xact_sys_tables	Non
pg_stat_xact_user_functions	Non
pg_stat_xact_user_tables	Non

Nom	Applicable à Aurora DSQL
pg_statio_sys_indexes	Non
pg_statio_sys_sequences	Non
pg_statio_sys_tables	Non
pg_statio_user_indexes	Non

La vue sys.jobs

sys.jobs fournit des informations sur le statut des tâches asynchrones. Par exemple, après avoir créé un index asynchrone, Aurora DSQL renvoie job_uuid. Vous pouvez utiliser job_uuid avec sys.jobs pour consulter le statut de la tâche.

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

Aurora DSQL renvoie une réponse semblable à ce qui suit.

```
           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
```

Le tableau suivant décrit les colonnes de la sys.jobs vue.

colonnes d'affichage sys.jobs

Colonne	Type	Description
job_id	text	Un UUID en base 32 représentant la tâche.

Colonne	Type	Description
`status`	`text`	État actuel de la tâche. Les valeurs possibles sont `submitted` , `processing` , `completed` et `failed`. Pour de plus amples informations, veuillez consulter valeurs de statut sys.jobs.
`details`	`text`	Tous les détails pertinents concernant le poste. Si la tâche échoue, une raison détaillée est fournie.
`job_type`	`text`	Type de tâche asynchrone. Les valeurs possibles sont les suivantes : `INDEX_BUILD` — une construction d'index asynchrone. `ANALYZE`— une tâche d'analyse automatique soumise par le système. `DROP`— supprime les données physiques après une `DROP INDEX` opération `DROP TABLE` OR.
`class_id`	`oid`	L'OID de la table de catalogue qui contient l'objet.
`object_id`	`oid`	L'OID de l'objet.
`object_name`	`text`	Nom complet de l'objet. `DROP`les tâches ne peuvent pas référencer des objets déjà déposés. Si un objet référencé a déjà été supprimé, il `object_name` peut être NULL.
`start_time`	`timestamp with time zone`	Horodatage auquel le travail a été soumis.
`update_time`	`timestamp with time zone`	Horodatage auquel la ligne de tâche a été mise à jour pour la dernière fois.

valeurs de statut sys.jobs

Statut	Description
`submitted`	La tâche est envoyée, mais Aurora DSQL n'a pas encore commencé à la traiter.
`processing`	Aurora DSQL est en train de traiter la tâche.
`failed`	La tâche a échoué. Consultez la `details` colonne pour plus d'informa tions.
`completed`	Aurora DSQL a terminé la tâche avec succès.

La vue sys.iam_pg_role_mappings

La vue `sys.iam_pg_role_mappings` fournit des informations sur les autorisations accordées aux utilisateurs IAM. Par exemple, si `DQSLDBConnect` est un rôle IAM qui donne accès à Aurora DSQL à des non-administrateurs et qu'un utilisateur nommé `testuser` reçoit le rôle `DQSLDBConnect` et les autorisations correspondantes, vous pouvez interroger la vue `sys.iam_pg_role_mappings` pour voir quels utilisateurs ont obtenu quelles autorisations.

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

Requêtes utiles sur les métadonnées du système

Utilisez ces requêtes pour obtenir les statistiques des tables et les métadonnées du système sans effectuer d'opérations coûteuses telles que l'analyse complète des tables.

Obtenir une estimation du nombre de lignes pour un tableau

Pour obtenir le nombre approximatif de lignes d'une table sans effectuer une analyse complète de la table, utilisez la requête suivante :

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

La commande renvoie un résultat semblable à ce qui suit :

```
  reltuples
```

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

Cette approche est plus efficace que `SELECT COUNT(*)` pour les grandes tables dans Aurora DSQL.

Obtenir la version majeure actuelle d'Aurora DSQL

Pour obtenir la version majeure actuelle du cluster Aurora DSQL, utilisez la requête suivante :

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

La commande renvoie un résultat semblable à ce qui suit :

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

Cela renvoie la version principale sur laquelle se trouve la connexion SQL dans Aurora DSQL.

Obtenir la version actuelle de PostgreSQL

Pour obtenir la version actuelle de PostgreSQL du cluster Aurora DSQL, utilisez la requête suivante :

```
SHOW server_version;
```

La commande renvoie un résultat semblable à ce qui suit :

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

Cela renvoie la version de PostgreSQL utilisée par la connexion SQL dans Aurora DSQL.

Commande **ANALYZE**.

La commande `ANALYZE` collecte des statistiques sur le contenu des tables de la base de données et enregistre les résultats dans la vue système `pg_stats`. Le planificateur de requêtes utilise ensuite ces statistiques pour déterminer les plans d'exécution les plus efficaces pour les requêtes.

Dans Aurora DSQL, vous ne pouvez pas exécuter la commande ANALYZE dans le cadre d'une transaction explicite. ANALYZE n'est pas soumis au délai d'expiration des transactions de base de données.

Pour réduire le besoin d'intervention manuelle et maintenir les statistiques constamment à jour, Aurora DSQL exécute automatiquement ANALYZE en tant que processus d'arrière-plan. Cette tâche en arrière-plan est déclenchée automatiquement en fonction du taux de variation observé dans le tableau. Il est lié au nombre de lignes (tuples) qui ont été insérées, mises à jour ou supprimées depuis la dernière analyse.

ANALYZE s'exécute de manière asynchrone en arrière-plan et son activité peut être surveillée dans la vue système sys.jobs avec la requête suivante :

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

Considérations clés

> **Note**
>
> Les tâches ANALYZE sont facturées comme les autres tâches asynchrones dans Aurora DSQL. Lorsque vous modifiez une table, cela peut déclencher indirectement une tâche automatique de collecte de statistiques en arrière-plan, ce qui peut entraîner des frais de comptage en raison de l'activité associée au niveau du système.

Les tâches ANALYZE en arrière-plan, déclenchées automatiquement, collectent les mêmes types de statistiques qu'une ANALYZE manuelle et les appliquent par défaut aux tables utilisateur. Les tables du système et du catalogue sont exclues de ce processus automatisé.

Utilisation des plans Aurora SQL EXPLAIN

Aurora DSQL utilise une structure de plan EXPLAIN similaire à celle de PostgreSQL, mais avec des ajouts clés qui reflètent son architecture distribuée et son modèle d'exécution.

Dans cette documentation, nous allons fournir un aperçu des plans Aurora DSQL EXPLAIN, en mettant en évidence les similitudes et les différences par rapport à PostgreSQL. Nous aborderons les différents types d'opérations de scan disponibles dans Aurora DSQL et vous aiderons à comprendre le coût de l'exécution de vos requêtes.

Plans PostgreSQL VS Aurora DSQL EXPLAIN

Aurora DSQL repose sur la base de données PostgreSQL et partage la plupart des structures de plan avec PostgreSQL, mais présente des différences architecturales majeures qui ont une incidence sur l'exécution et l'optimisation des requêtes :

Fonctionnalité	PostgreSQL	Aurora DSQL
Stockage des données	Stockage en tas	Pas de tas, toutes les lignes sont indexées par un identifiant unique
Clé primaire	L'index de clé primaire est distinct des données de table	L'index de clé primaire est la table avec toutes les colonnes supplémentaires sous forme de colonnes INCLUDE
Index secondaires	Index secondaires standard	Fonctionne de la même manière que PostgreSQL, avec la possibili té d'inclure des colonnes non clés
Capacités de filtrage	État de l'index, filtre en tas	État de l'index, filtre de stockage, filtre du processeur de requêtes
Types d’analyse	Numérisation séquentielle, analyse de l'index, analyse de l'index uniquement	Scan complet, scan de l'index uniquement, scan de l'index
Exécution de requêtes	Local dans la base de données	Distribué (le calcul et le stockage sont séparés)

Aurora DSQL stocke les données des tables directement dans l'ordre des clés primaires plutôt que dans un tas séparé. Chaque ligne est identifiée par une clé unique, généralement la clé primaire, qui permet à la base de données d'optimiser les recherches de manière plus efficace. Cette différence architecturale explique pourquoi Aurora DSQL utilise souvent des scans d'index uniquement dans les cas où PostgreSQL choisit un scan séquentiel.

Autre distinction essentielle : Aurora DSQL sépare le calcul du stockage, ce qui permet d'appliquer des filtres plus tôt dans le processus d'exécution afin de réduire le mouvement des données et d'améliorer les performances.

Pour en savoir plus sur l'utilisation des plans EXPLAIN avec PostgreSQL, consultez la documentation de PostgreSQL EXPLAIN.

Éléments clés des plans Aurora SQL EXPLAIN

Les plans Aurora DSQL EXPLAIN fournissent des informations détaillées sur la manière dont les requêtes sont exécutées, notamment sur l'emplacement du filtrage et sur les colonnes extraites du stockage. La compréhension de ce résultat vous permet d'optimiser les performances des requêtes.

Indice Condo

: Conditions utilisées pour naviguer dans l'index. Filtrage le plus efficace qui réduit les données numérisées. Dans Aurora DSQL, les conditions d'index peuvent être appliquées à plusieurs couches du plan d'exécution.

Projections

: Colonnes extraites du stockage. Moins de projections sont synonymes de meilleures performances.

Filtre de stockage

: Conditions appliquées au niveau du stockage. Plus efficace que les filtres du processeur de requêtes.

Filtre du processeur de requêtes

: Conditions appliquées au niveau du processeur de requêtes. Nécessite le transfert de toutes les données avant le filtrage, ce qui entraîne une augmentation des mouvements de données et une surcharge de traitement.

Filtres dans Aurora DSQL

Aurora DSQL sépare le calcul du stockage, ce qui signifie que le moment où les filtres sont appliqués pendant l'exécution des requêtes a un impact significatif sur les performances. Les filtres appliqués avant le transfert de gros volumes de données réduisent la latence et améliorent l'efficacité. Plus un filtre est appliqué tôt, moins les données doivent être traitées, déplacées et numérisées, ce qui accélère les requêtes.

Aurora DSQL peut appliquer des filtres à plusieurs étapes du chemin de requête. Il est essentiel de comprendre ces étapes pour interpréter les plans de requêtes et optimiser les performances.

Niveau	Type de filtre	Description
1	État de l'indice	Appliqué lors de la numérisation de l'index. Limite la quantité de données lues depuis le stockage et réduit le nombre de données envoyées à la couche de calcul.
2	Filtre de stockage	Appliqué après la lecture des données depuis le stockage, mais avant leur envoi au calcul. Voici un exemple de filtre sur une colonne d'inclusion d'un index. Réduit le transfert de données, mais pas la quantité lue.
3	Filtre du processeur de requêtes	Appliqué une fois que les données ont atteint la couche de calcul. Toutes les données doivent d'abord être transférées, ce qui augmente le temps de latence et les coûts. À l'heure actuelle, Aurora DSQL ne peut pas effectuer toutes les opération s de filtrage et de projection sur le stockage. Certaines requêtes peuvent donc être obligées de recourir à ce type de filtrage.

Lecture des plans Aurora SQL EXPLAIN

Comprendre comment lire les plans EXPLAIN est essentiel pour optimiser les performances des requêtes. Dans cette section, nous allons passer en revue des exemples concrets de plans de requêtes SQL Aurora, montrer le comportement des différents types de scan, expliquer où les filtres sont appliqués et mettre en évidence les opportunités d'optimisation.

Exemples de tableaux utilisés dans ces exemples

Les exemples ci-dessous font référence à deux tableaux : `transaction` et`account`.

La `transaction` table ne possède pas de clé primaire, ce qui oblige Aurora DSQL à effectuer des analyses complètes de la table lorsqu'elle l'interroge.

La `account` table contient un index activé`customer_id`. Cet index inclut `balance` et `status` en tant que colonnes de couverture, ce qui permet de répondre à certaines requêtes directement à partir de l'index sans avoir à lire dans la table de base. Toutefois, l'index n'inclut pas`created_at`, de sorte que les requêtes qui font référence à cette colonne nécessitent un accès supplémentaire à la table.

```
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);
```

Exemple de scan complet

Aurora DSQL propose à la fois des scans séquentiels, qui sont fonctionnellement identiques à PostgreSQL, ainsi que des scans complets. La seule différence entre les deux est que les scans complets peuvent utiliser un filtrage supplémentaire sur le stockage. Pour cette raison, il est presque toujours sélectionné au-dessus des scans séquentiels. En raison de la similitude, nous ne couvrirons que des exemples des scans complets les plus intéressants.

Les scans complets seront principalement utilisés sur des tables sans clé primaire. Étant donné que les clés primaires Aurora DSQL sont par défaut des index de couverture complets, Aurora DSQL utilisera très probablement des scans d'index uniquement sur la clé primaire dans de nombreuses situations où PostgreSQL utiliserait un scan séquentiel. Comme c'est le cas pour la plupart des autres bases de données, une table dépourvue d'index risque d'être mal dimensionnée.

```
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)
```

Ce plan montre deux filtres appliqués à différentes étapes. La `transaction_date > '2025-01-01'` condition est appliquée au niveau de la couche de stockage, ce qui réduit la quantité de données renvoyées. La `description LIKE '%external%'` condition est appliquée ultérieurement dans le processeur de requêtes, une fois les données transférées, ce qui le rend moins efficace. L'introduction de filtres plus sélectifs dans les couches de stockage ou d'index améliore généralement les performances.

Exemple de scan avec index uniquement

Les scans d'index uniquement sont les types de scan les plus optimaux dans Aurora DSQL, car ils permettent de réduire le nombre d'allers-retours vers la couche de stockage et sont ceux qui peuvent effectuer le plus de filtrage. Mais ce n'est pas parce que vous voyez Index Only Scan que vous avez le meilleur plan. En raison des différents niveaux de filtrage qui peuvent se produire, il est essentiel de toujours faire attention aux différents endroits où le filtrage peut se produire.

```
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)
```

Dans ce plan, la condition de l'index `customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'` () est d'abord évaluée lors de l'analyse de l'index, qui est l'étape la plus efficace car elle limite la quantité de données lues depuis le stockage. Le filtre de stockage est appliqué après la lecture des données`status = 'pending'`, mais avant leur envoi à la couche de calcul, ce qui réduit la quantité de données transférées. Enfin, le filtre du processeur de requêtes s'exécute en dernier, une fois les données déplacées, ce qui le rend le moins efficace. `balance > 100` Parmi celles-ci, la condition d'index fournit les meilleures performances car elle contrôle directement la quantité de données numérisées.

Exemple de scan d'index

Les scans d'index sont similaires aux scans d'index uniquement, sauf qu'ils comportent l'étape supplémentaire d'appel à la table de base. Comme Aurora DSQL peut spécifier des filtres de stockage, il est en mesure de le faire à la fois lors de l'appel d'index et lors de l'appel de recherche.

Pour que cela soit clair, Aurora DSQL présente le plan sous la forme de deux nœuds. De cette façon, vous pouvez clairement voir dans quelle mesure l'ajout d'une colonne d'inclusion sera utile en termes de lignes renvoyées par le stockage.

```
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)
```

Ce plan montre comment le filtrage s'effectue en plusieurs étapes :

- La condition d'index sur les `customer_id` filtres permet de filtrer les données à un stade précoce.
- Le filtre de stockage permet de `status` restreindre davantage les résultats avant qu'ils ne soient envoyés au calcul.
- Le filtre du processeur de requêtes activé `balance` est appliqué ultérieurement, après le transfert.
- Le filtre de recherche activé `created_at` est évalué lors de la récupération de colonnes supplémentaires dans la table de base.

L'ajout de colonnes fréquemment utilisées en tant que `INCLUDE` champs permet souvent d'éliminer cette recherche et d'améliorer les performances.

Bonnes pratiques

- Alignez les filtres sur les colonnes indexées pour accélérer le filtrage.
- Utilisez les colonnes INCLUDE pour autoriser les scans d'index uniquement et éviter les recherches.
- Validez les estimations de lignes lorsque vous étudiez les problèmes de performances. Aurora DSQL gère automatiquement les statistiques `ANALYZE` en s'exécutant en arrière-plan en fonction des taux de modification des données. Si les estimations semblent inexactes, vous pouvez les exécuter `ANALYZE` manuellement pour actualiser immédiatement les statistiques.
- Évitez les requêtes non indexées sur de grandes tables afin d'éviter des scans complets coûteux.

Comprendre DPUs dans EXPLAIN ANALYZE

Aurora DSQL fournit des informations sur les unités de traitement distribué (DPU) au niveau des instructions dans les résultats du `EXPLAIN ANALYZE VERBOSE` plan, ce qui vous donne une meilleure visibilité sur le coût des requêtes pendant le développement. Cette section explique ce DPUs que c'est et comment les interpréter dans le `EXPLAIN ANALYZE VERBOSE` résultat.

Qu'est-ce qu'un DPU ?

Une unité de traitement distribuée (DPU) est la mesure normalisée du travail effectué par Aurora DSQL. Il est composé de :

- ComputedPU — Temps passé à exécuter des requêtes SQL
- ReadDPU — Ressources utilisées pour lire les données depuis le stockage

- WriteDPU - Ressources utilisées pour écrire des données dans le stockage
- MultiRegionWriteDPU : ressources utilisées pour répliquer les écritures sur des clusters homologues dans des configurations multirégionales.

Utilisation du DPU dans EXPLAIN ANALYZE VERBOSE

Aurora DSQL s'étend `EXPLAIN ANALYZE VERBOSE` pour inclure une estimation de l'utilisation du DPU au niveau des instructions jusqu'à la fin de la sortie. Cela fournit une visibilité immédiate sur le coût des requêtes, vous aidant à identifier les facteurs de coût de la charge de travail, à optimiser les performances des requêtes et à mieux prévoir l'utilisation des ressources.

Les exemples suivants montrent comment interpréter les estimations du DPU au niveau des instructions incluses dans la sortie EXPLAIN ANALYZE VERBOSE.

Exemple 1 : requête SELECT

```
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
```

Dans cet exemple, l'instruction SELECT effectue une analyse d'index uniquement. La majeure partie du coût provient donc de Read DPU (0,04312), qui représente les données extraites du stockage, et

de Compute DPU (0,01607), qui reflète les ressources de calcul utilisées pour traiter et renvoyer les résultats. Il n'y a pas de DPU d'écriture puisque la requête ne modifie pas les données. Le DPU total (0,05919) est la somme de Compute + Read + Write.

Exemple 2 : requête INSERT

```
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
```

Cette instruction effectue principalement des écritures, de sorte que la majeure partie du coût est associée au DPU d'écriture. Le Compute DPU (0,01550) représente le travail effectué pour traiter et insérer les valeurs. Le DPU de lecture (0,00307) reflète les lectures mineures du système (pour les recherches de catalogue ou les vérifications d'index).

Notez les minimums de transaction indiqués à côté de Lecture et écriture DPUs. Ils indiquent les coûts de base par transaction qui s'appliquent uniquement lorsque l'opération inclut des lectures ou des écritures. Cela ne signifie pas que chaque transaction entraîne automatiquement une charge de 0,00375 DPU en lecture ou de 0,05 en écriture DPU. Ces minimums sont plutôt appliqués au niveau de la transaction lors de l'agrégation des coûts et uniquement si des lectures ou des écritures ont lieu dans le cadre de cette transaction. En raison de cette différence de portée, les estimations au niveau des relevés `EXPLAIN ANALYZE VERBOSE` peuvent ne pas correspondre exactement aux mesures au niveau des transactions indiquées dans les données de facturation. CloudWatch

Utilisation des informations du DPU pour l'optimisation

Les estimations du DPU par instruction vous offrent un moyen puissant d'optimiser les requêtes au-delà du simple temps d'exécution. Cas d'utilisation courants :

- Connaissance des coûts : déterminez le coût d'une requête par rapport aux autres.
- Optimisation du schéma : comparez l'impact des index ou des modifications de schéma sur les performances et l'efficacité des ressources.
- Planification du budget : estimez le coût de la charge de travail en fonction de l'utilisation observée du DPU.
- Comparaison des requêtes : évaluez les approches de requête alternatives en fonction de leur consommation relative de DPU.

Interprétation des informations du DPU

Tenez compte des meilleures pratiques suivantes lorsque vous utilisez des données DPU provenant de `EXPLAIN ANALYZE VERBOSE` :

- Utilisez-le de manière directionnelle : considérez le DPU indiqué comme un moyen de comprendre le coût relatif d'une requête plutôt que comme une correspondance exacte avec les CloudWatch métriques ou les données de facturation. Des différences sont attendues car les `EXPLAIN ANALYZE VERBOSE` rapports indiquent les coûts au niveau des relevés, tandis qu'ils CloudWatch regroupent l'activité au niveau des transactions. CloudWatch inclut également les opérations en arrière-plan (telles que ANALYZE ou les compactages) et les frais généraux de transaction (`BEGIN`/`COMMIT`) qui sont `EXPLAIN ANALYZE VERBOSE` intentionnellement exclus.
- La variabilité du DPU entre les cycles est normale dans les systèmes distribués et n'indique aucune erreur. Des facteurs tels que la mise en cache, les modifications du plan d'exécution, la simultanéité ou les changements dans la distribution des données peuvent tous entraîner la consommation de ressources différentes d'une requête à l'autre.
- Petites opérations par lots : si votre charge de travail génère de nombreux petits relevés, envisagez de les regrouper en opérations plus importantes (ne dépassant pas 10 Mo). Cela permet de réduire les frais généraux d'arrondissement et de produire des estimations de coûts plus pertinentes.
- À utiliser pour le réglage, pas pour la facturation : les données DPU entrées `EXPLAIN ANALYZE VERBOSE` sont conçues pour la prise en compte des coûts, le réglage des requêtes et l'optimisation. Il ne s'agit pas d'un indicateur de facturation. Fiez-vous toujours aux CloudWatch

indicateurs ou aux rapports de facturation mensuels pour obtenir des données fiables sur les coûts et l'utilisation.

Gestion des clusters Aurora DSQL

Aurora DSQL propose plusieurs options de configuration pour vous aider à établir l'infrastructure de base de données adaptée à vos besoins. Pour configurer votre infrastructure de cluster Aurora DSQL, consultez les sections suivantes.

Rubriques

- Configuration de clusters à une seule région
- Configuration de clusters multi-régions
- Configuration de clusters Aurora DSQL à l'aide d'AWS CloudFormation
- Cycle de vie du cluster Aurora SQL

Les caractéristiques et fonctionnalités décrites dans ce guide garantissent que votre environnement Aurora DSQL est plus résilient, réactif et capable de prendre en charge vos applications à mesure de leur croissance et de leur évolution.

Configuration de clusters à une seule région

Configurez et gérez des clusters pour une Région AWS à l'aide de l'AWS CLI ou de votre langage de programmation préféré, notamment Python, C++, JavaScript, Java, Rust, Ruby, .NET et Golang. L'AWS CLI fournit un accès rapide via des commandes shell, tandis que les kits de développement logiciel (SDK) AWS permettent un contrôle programmatique grâce à la prise en charge du langage natif.

Rubriques

- Utilisation des kits SDK AWS
- Utilisation de l'interface de ligne de commande (CLI) AWS

Utilisation des kits SDK AWS

Les kitsAWS SDK fournissent un accès programmatique à Aurora DSQL dans votre langage de programmation préféré. Les sections suivantes montrent comment effectuer des opérations de cluster courantes à l'aide de différents langages de programmation.

Créer un cluster

Les exemples suivants montrent comment créer un cluster à une seule région à l'aide de différents langages de programmation.

Python

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
import boto3


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

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

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


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


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

C++

L'exemple suivant vous permet de créer un cluster à une seule Région AWS.

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

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

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

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

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

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

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

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

    return cluster;
}

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

            auto cluster = CreateCluster(region);

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

JavaScript

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Utilisez l'exemple suivant pour créer un cluster en une seule Région AWS.

```
package org.example;

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

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

public class CreateCluster {

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

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

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

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

Rust

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour créer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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)
 }

}
```

Obtention d'un cluster

Les exemples suivants montrent comment obtenir des informations sur un cluster à une seule région à l'aide de différents langages de programmation.

Python

Pour obtenir des informations sur un cluster à une seule région, utilisez l'exemple suivant.

```
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++

Utilisez l'exemple suivant pour obtenir des informations sur un cluster à une seule région.

```
#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

Pour obtenir des informations sur un cluster à une seule région, utilisez l'exemple suivant.

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

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

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

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

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

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

main();
```

Java

L'exemple suivant vous permet d'obtenir des informations sur un cluster à une seule région.

```
package org.example;

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

public class GetCluster {

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

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

Rust

L'exemple suivant vous permet d'obtenir des informations sur un cluster à une seule région.

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

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

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

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

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

    Client::from_conf(config)
}

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

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

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

    Ok(())
}
```

Ruby

L'exemple suivant vous permet d'obtenir des informations sur un cluster à une seule région.

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

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

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

main if $PROGRAM_NAME == __FILE__
```

.NET

L'exemple suivant vous permet d'obtenir des informations sur un cluster à une seule région.

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

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

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

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

                return await client.GetClusterAsync(getClusterRequest);
            }
        }

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

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

Golang

L'exemple suivant vous permet d'obtenir des informations sur un cluster à une seule région.

```
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)
 }
}
```

Mise à jour d'un cluster

Les exemples suivants montrent comment mettre à jour un cluster à une seule région à l'aide de différents langages de programmation.

Python

Pour créer un cluster à une seule région, utilisez l'exemple suivant.

```
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++

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
#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

Pour créer un cluster à une seule région, utilisez l'exemple suivant.

```
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

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
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

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
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

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
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

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
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

Utilisez l'exemple suivant pour créer un cluster à une seule région.

```
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)
 }
}
```

Suppression d'un cluster

Les exemples suivants montrent comment supprimer un cluster à une seule région à l'aide de différents langages de programmation.

Python

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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++

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
#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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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

Pour supprimer un cluster à une seule Région AWS, utilisez l'exemple suivant.

```
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)
 }
```

```
}
```

Pour plus d'exemples de code, visitez le référentiel GitHub d'exemples Aurora DSQL.

Utilisation de l'interface de ligne de commande (CLI) AWS

L'interface de ligne de commande (CLI) AWS fournit une interface de ligne de commande pour gérer vos clusters Aurora DSQL. L'exemple suivant montre les opérations courantes de gestion de cluster.

Créer un cluster

Créez un cluster à l'aide de la commande create-cluster.

> **Note**
>
> La création de clusters est une opération asynchrone. Appelez l'API GetCluster jusqu'à ce que le statut passe à `ACTIVE`. Vous pouvez vous connecter à votre cluster une fois qu'il est actif.

Example Commande

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

> **Note**
>
> Pour désactiver la protection contre la suppression lors de la création, incluez l'indicateur `--no-deletion-protection-enabled`.

Example Réponse

```
{
    "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"
    }
}
```

Description d'un cluster

Obtenez des informations sur un cluster à l'aide de la commande get-cluster.

Example Commande

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

Example Réponse

```
{
    "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"
    }
}
```

Mise à jour d'un cluster

Mettez à jour un cluster existant à l'aide de la commande update-cluster.

> **Note**
>
> Les mises à jour sont des opérations asynchrones. Appelez l'API GetCluster jusqu'à ce que le statut passe à `ACTIVE` pour voir vos modifications.

Example Commande

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

Example Réponse

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

Suppression d'un cluster

Supprimez un cluster existant à l'aide de la commande delete-cluster.

> **Note**
>
> Vous pouvez uniquement supprimer les clusters dont la protection contre la suppression est désactivée. La protection contre la suppression est activée par défaut lorsque vous créez de nouveaux clusters.

Example Commande

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

Example Réponse

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
```

```
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

Établissement de la liste des clusters

Répertoriez vos clusters à l'aide de la commande list-clusters.

Example Commande

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

Example Réponse

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

Configuration de clusters multi-régions

Configurez et gérez des clusters dans plusieurs Régions AWS à l'aide de l'AWS CLI ou de votre langage de programmation préféré, notamment Python, C++, JavaScript, Java, Rust, Ruby, .NET et Golang. L'AWS CLI fournit un accès rapide via des commandes shell, tandis que les kits SDK AWS permettent un contrôle programmatique grâce à la prise en charge du langage natif.

Rubriques

- Utilisation des kits SDK AWS
- Utilisation de la AWS CLI

Utilisation des kits SDK AWS

Les kitsAWS SDK fournissent un accès programmatique à Aurora DSQL dans votre langage de programmation préféré. Les sections suivantes montrent comment effectuer des opérations de cluster courantes à l'aide de différents langages de programmation.

Créer un cluster

Les exemples suivants montrent comment créer un cluster multi-régions à l'aide de différents langages de programmation.

Python

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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++

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
#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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

```
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

Pour créer un cluster multi-régions, utilisez l'exemple suivant. La création d'un cluster multi-régions peut prendre du temps.

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

Obtention d'un cluster

Les exemples suivants montrent comment obtenir des informations sur un cluster multi-régions à l'aide de différents langages de programmation.

Python

Pour obtenir des informations sur un cluster multi-régions, utilisez l'exemple suivant.

```
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++

Utilisez l'exemple suivant pour obtenir des informations sur un cluster multi-régions.

```
#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

Pour obtenir des informations sur un cluster multi-régions, utilisez l'exemple suivant.

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

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

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

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

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

  const response = await getCluster(region, clusterId);
```

```
  console.log("Cluster: ", response);
}

main();
```

Java

L'exemple suivant vous permet d'obtenir des informations sur un cluster multi-régions.

```
package org.example;

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

public class GetCluster {

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

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

Rust

L'exemple suivant vous permet d'obtenir des informations sur un cluster multi-régions.

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

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

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

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

    Client::from_conf(config)
}

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

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

    let cluster = get_cluster(region, "<your cluster id>").await;
```

```
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

L'exemple suivant vous permet d'obtenir des informations sur un cluster multi-régions.

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

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

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

main if $PROGRAM_NAME == __FILE__
```

.NET

L'exemple suivant vous permet d'obtenir des informations sur un cluster multi-régions.

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

namespace DSQLExamples.examples
{
```

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

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

               return await client.GetClusterAsync(getClusterRequest);
           }
       }

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

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

```
}
```

Golang

L'exemple suivant vous permet d'obtenir des informations sur un cluster multi-régions.

```
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)
 }
}
```

Mise à jour d'un cluster

Les exemples suivants montrent comment mettre à jour un cluster multi-régions à l'aide de différents langages de programmation.

Python

Pour mettre à jour un cluster multi-régions, utilisez l'exemple suivant.

```
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++

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
#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

Pour mettre à jour un cluster multi-régions, utilisez l'exemple suivant.

```
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

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
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

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
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

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
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

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
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

Utilisez l'exemple suivant pour mettre à jour un cluster multi-régions.

```
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)
 }
}
```

Suppression d'un cluster

Les exemples suivants montrent comment supprimer un cluster multi-régions à l'aide de différents langages de programmation.

Python

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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++

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
#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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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

Pour supprimer un cluster multi-régions, utilisez l'exemple suivant. La suppression d'un cluster multi-régions peut prendre du temps.

```
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)
 }
}
```

Pour plus d'exemples de code, visitez le référentiel GitHub d'exemples Aurora DSQL.

Utilisation de la AWS CLI

La AWS CLI fournit une interface de ligne de commande pour gérer vos clusters Aurora DSQL multirégionaux. Les exemples suivants montrent comment créer, configurer et supprimer des clusters multi-régions.

Connexion à votre cluster multi-régions

Les clusters multi-régions associés fournissent deux points de terminaison régionaux, un dans chaque cluster associé Région AWS. Les deux points de terminaison présentent une base de données logique unique qui prend en charge les opérations de lecture et d'écriture simultanées avec une forte cohérence des données. Outre les clusters associés, un cluster multi-régions possède également une région témoin qui stocke une fenêtre limitée de journaux de transactions chiffrés, qui est utilisée pour améliorer la durabilité et la disponibilité de plusieurs régions. Les régions témoins multi-régions n'ont pas de points de terminaison.

Création de clusters multi-régions

Pour créer des clusters multi-régions, vous devez d'abord créer un cluster avec une région témoin. Ensuite, vous associez ce cluster à un second cluster qui partage la même région témoin que votre premier cluster. L'exemple suivant montre comment créer des clusters dans les régions USA Est (Virginie du Nord) et USA Est (Ohio) avec la région USA Ouest (Oregon) comme région témoin.

Étape 1 : création du cluster 1 dans la région USA Est (Virginie du Nord)

Pour créer un cluster dans l'est des États-Unis (Virginie du Nord) Région AWS avec des propriétés multirégionales, utilisez la commande ci-dessous.

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

Example Réponse :

```
{
    "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**
>
> Lorsque l'opération d'API aboutit, le cluster passe à l'état `PENDING_SETUP`. La création du cluster reste à l'état `PENDING_SETUP` jusqu'à ce que vous le mettiez à jour avec l'ARN de son cluster homologue.

Étape 2 : création du cluster 2 dans la région USA Est (Ohio)

Pour créer un cluster dans l'est des États-Unis (Ohio) Région AWS avec des propriétés multirégionales, utilisez la commande ci-dessous.

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

Example Réponse :

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

Lorsque l'opération d'API réussit, le cluster passe à l'état `PENDING_SETUP`. La création du cluster reste à l'état `PENDING_SETUP` jusqu'à ce que vous le mettiez à jour avec l'ARN d'un autre cluster associé.

Étape 3 : association d'un cluster de la région USA Est (Virginie du Nord) avec la région USA Est (Ohio)

Pour associer votre cluster de la région USA Est (Virginie du Nord) à votre cluster de la région USA Est (Ohio), utilisez la commande `update-cluster`. Spécifiez le nom de votre cluster de la région USA Est (Virginie du Nord) et une chaîne JSON avec l'ARN du cluster de la région USA Est (Ohio).

```
aws dsql update-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4' \
--multi-region-properties '{"witnessRegion": "us-west-2","clusters": ["arn:aws:dsql:us-
east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"]}'
```

Example Réponse

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

Étape 4 : association d'un cluster de la région USA Est (Ohio) avec la région USA Est (Virginie du Nord)

Pour associer votre cluster de la région USA Est (Ohio) à votre cluster de la région USA Est (Virginie du Nord), utilisez la commande `update-cluster`. Spécifiez le nom de votre cluster de la région USA Est (Ohio) et une chaîne JSON avec l'ARN du cluster de la région USA Est (Virginie du Nord).

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 Réponse

```
{
```

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

> **Note**
>
> Après une association réussie, les deux clusters passent de l'état « PENDING_SETUP » à « CREATING » et enfin à l'état « ACTIVE » lorsqu'ils sont prêts à être utilisés.

Affichage des propriétés d'un cluster multi-régions

Lorsque vous décrivez un cluster, vous pouvez afficher les propriétés multirégionales des clusters de différentes Régions AWS régions.

Example

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

Example Réponse

```
{
    "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"
       ]
```

```
    }
}
```

Association des clusters lors de la création

Vous pouvez réduire le nombre d'étapes en incluant des informations d'association lors de la création du cluster. Après avoir créé votre premier cluster dans la région USA Est (Virginie du Nord) (étape 1), vous pouvez créer votre deuxième cluster dans la région USA Est (Ohio) tout en lançant le processus d'association en incluant l'ARN du premier cluster.

Example

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2","clusters": ["arn:aws:dsql:us-
east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Cela combine les étapes 2 et 4, mais vous devez tout de même terminer l'étape 3 (mise à jour du premier cluster avec l'ARN du deuxième cluster) pour établir la relation d'association. Une fois toutes les étapes terminées, les deux clusters passeront par les mêmes états que dans le processus standard : de PENDING_SETUP à CREATING, puis à ACTIVE lorsqu'ils seront prêts à être utilisés.

Suppression de clusters multi-régions

Pour supprimer un cluster multi-régions, vous devez effectuer deux étapes.

1. Désactivez la protection contre la suppression pour chaque cluster.
2. Supprimez chaque cluster pair séparément dans ses clusters respectifs Région AWS

Mise à jour et suppression d'un cluster dans la région USA Est (Virginie du Nord)

1. Désactivez la protection contre la suppression à l'aide de la commande `update-cluster`.

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

2. Supprimez le cluster à l'aide de la commande `delete-cluster`.

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

Cette commande renvoie la réponse suivante.

```
{
    "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**
>
> Le cluster passe à l'état `PENDING_DELETE`. La suppression n'est pas terminée tant que vous n'avez pas supprimé le cluster associé dans la région USA Est (Ohio).

Mise à jour et suppression d'un cluster dans la région USA Est (Ohio)

1. Désactivez la protection contre la suppression à l'aide de la commande `update-cluster`.

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

2. Supprimez le cluster à l'aide de la commande `delete-cluster`.

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

Cette commande renvoie la réponse suivante :

```
{
    "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**
>
> Le cluster passe à l'état `PENDING_DELETE`. Après quelques secondes, le système fait passer automatiquement les deux clusters associés à l'état `DELETING` après validation.

Configuration de clusters Aurora DSQL à l'aide d'AWS CloudFormation

Vous pouvez utiliser la même ressource CloudFormation `AWS::DSQL::Cluster` pour déployer et gérer des clusters Aurora DSQL à une seule région ou multi-régions.

Consultez la référence du type de ressource Amazon Aurora DSQL pour en savoir plus sur la création, la modification et la gestion de clusters à l'aide de la ressource `AWS::DSQL::Cluster`.

Création de la configuration de cluster initiale

Créez d'abord un modèle AWS CloudFormation pour définir votre cluster multi-régions :

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

Créez des piles dans les deux régions à l'aide des commandes de l'interface de ligne de commande (CLI) AWS suivantes :

```
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
```

Recherche des identifiants de cluster

Récupérez les identifiants de ressources physiques pour vos clusters :

```
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"
]
```

Mise à jour de la configuration d'un cluster

Mettez à jour votre modèle AWS CloudFormation pour inclure les deux ARN du cluster :

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
        Clusters:
        - arn:aws:dsql:us-east-2:123456789012:cluster/auabudrks5jwh4mjt6o5xxhr4y
        - arn:aws:dsql:us-east-1:123456789012:cluster/imabudrfon4p2z3nv2jo4rlajm
```

Appliquez la configuration mise à jour aux deux régions :

```
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
```

Cycle de vie du cluster Aurora SQL

Comprendre le cycle de vie des clusters Aurora DSQL vous permet de gérer efficacement vos clusters. Cette section couvre les définitions de l'état des clusters et la fonctionnalité d'échelle jusqu'à zéro qui optimise les coûts.

Définition de l'état du cluster Aurora DSQL

L'état du cluster Aurora DSQL fournit des informations essentielles sur l'état et la connectivité du cluster. Vous pouvez consulter l'état des clusters et des instances de cluster à l'aide de l'API SQL AWS Management Console AWS CLI, ou Aurora.

Le tableau suivant décrit chaque état possible pour un cluster Aurora DSQL et sa signification.

Statut	Description
Création	Aurora DSQL tente de créer ou de configurer des ressources pour le cluster. Toute tentative de connexion échouera tant qu'un cluster affiche cet état.
Actif	Le cluster est opérationnel et prêt à être utilisé.
Inactif	Un cluster devient inactif lorsqu'il reste inactif suffisamment longtemps pour qu'Aurora DSQL puisse réduire les ressources en cours afin de réduire la capacité et les coûts. Lorsque vous vous connectez à un cluster inactif, Aurora DSQL rétablit l'état actif du cluster.
Inactif	Un cluster inactif devient inactif lorsqu'il n'y a aucune activité sur le cluster pendant une période prolongée. Dans cet état suspendu, les ressources en

Statut	Description
	cours d'exécution sont réduites à zéro tandis que vos données sont préservées. Lorsque vous tentez de vous connecter à un cluster inactif, Aurora DSQL établit automatiquement l'état actif du cluster. Le délai de restauration dépend de la taille du cluster.
Mise à jour en cours	Un cluster passe à l'état Mise à jour lorsque vous modifiez la configuration du cluster.
Suppression en cours	Un cluster passe à l'état Suppression lorsque vous soumettez une demande de suppression.
Supprimé	Le cluster a été correctement supprimé.
Échec	Aurora DSQL n'a pas pu créer le cluster car il a rencontré une erreur.
En attente de configuration	Pour les clusters multi-régions uniquement. Un cluster multi-régions passe à l'état En attente de configuration lorsque vous créez un cluster multi-régions dans votre première région avec une région témoin. La création de clusters est interrompue jusqu'à ce que vous créiez un autre cluster dans une région secondaire et que vous associiez les deux clusters ensemble.
En attente de suppression	Pour les clusters multi-régions uniquement. Un cluster multi-régions passe à l'état En attente de suppression lorsque vous en supprimez un. Le cluster passe à l'état Suppression une fois que vous avez supprimé le dernier cluster homologue.

Utilisation de clusters inactifs et inactifs

Lorsqu'Aurora DSQL ne détecte aucune activité de connexion sur un cluster pendant un certain temps, il fait passer le cluster à l'état inactif, réduisant ainsi les ressources de fonctionnement afin de minimiser la capacité et les coûts. Si l'activité de connexion reste absente pendant une période prolongée, le cluster inactif passe automatiquement à l'état inactif, où les ressources en cours d'exécution sont réduites à zéro tout en préservant vos données.

Pour reprendre le fonctionnement normal, il suffit de se connecter au cluster comme d'habitude. Lorsque vous vous connectez avec succès au cluster, Aurora Aurora DSQL fait automatiquement passer le cluster à l'état actif.

> **Note**
>
> La première tentative de connexion à un cluster inactif ou inactif sera plus lente que d'habitude.

Opérations nécessitant un état de cluster actif

Certaines opérations nécessitent que votre cluster soit dans un état actif. Pour effectuer ces opérations sur un cluster inactif ou inactif, vous devez replacer votre cluster en mode actif en vous connectant à votre cluster.

Opérations de sauvegarde

L'exécution d'une sauvegarde nécessite un état de cluster actif. Si votre cluster est inactif ou inactif, les sauvegardes échouent avec le message d'erreur suivant :

```
"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."
}
```

Pour procéder à une sauvegarde, procédez comme suit :

1. Connectez-vous au cluster à l'aide de votre client de base de données préféré ou de la console Aurora DSQL pour le réveiller.
2. Attendez le passage automatique à l'état actif.
3. Lancez la sauvegarde une fois que le cluster est totalement opérationnel.

> **Note**
>
> Les sauvegardes existantes effectuées avant que le cluster ne soit inactif ou inactif restent valides et inchangées. Les nouvelles tentatives de sauvegarde sur le cluster échoueront jusqu'à ce que le cluster soit connecté pour un réveil automatique.

Affichage de l'état de votre cluster Aurora DSQL

Pour consulter l'état de votre cluster, utilisez l' AWS Management Console API SQL ou Aurora. AWS CLI

Console

Procédez comme suit pour afficher l'état du cluster dans la AWS Management Console :

Affichage de l'état du cluster dans la console

1. Ouvrez la console Aurora DSQL à l'adresse https://console.aws.amazon.com/dsql.
2. Choisissez Clusters dans le volet de navigation.
3. Consultez l'état de chaque cluster dans le tableau de bord.

AWS CLI

Utilisez la AWS CLI commande suivante pour vérifier l'état d'un seul cluster.

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

Exécutez la commande ci-dessous pour répertorier l'état de tous les clusters.

```
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
```

Cet exemple de sortie montre deux clusters actifs et un cluster en cours de suppression.

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

Programmation avec Aurora DSQL

Aurora DSQL met à votre disposition les outils suivants pour gérer vos ressources Aurora DSQL par programme.

AWS Command Line Interface (AWS CLI)

Vous pouvez créer et gérer vos ressources à l'aide de l' AWS CLI interface de ligne de commande. AWS CLI Fournit un accès direct au APIs for Services AWS, tel qu'Aurora DSQL. Pour obtenir la syntaxe et des exemples de commandes pour Aurora DSQL, consultez dsql dans la référence des commandes AWS CLI .

AWS kits de développement logiciel (SDKs)

AWS fournit SDKs de nombreuses technologies et langages de programmation populaires. Ils vous permettent d'appeler plus facilement Services AWS depuis vos applications dans ce langage ou cette technologie. Pour plus d'informations à ce sujet SDKs, consultez la section Outils de développement et de gestion d'applications sur AWS.

API Aurora DSQL

Cette API est une autre interface de programmation pour Aurora DSQL. Lorsque vous utilisez cette API, vous devez formater correctement chaque demande HTTPS et ajouter une signature numérique valide à chaque demande. Pour de plus amples informations, veuillez consulter Référence d'API.

CloudFormation

AWS::DSQL::ClusterIl s'agit d'une CloudFormation ressource qui vous permet de créer et de gérer des clusters Aurora DSQL dans le cadre de votre infrastructure en tant que code. CloudFormation vous aide à définir AWS l'ensemble de votre environnement dans le code, ce qui facilite le provisionnement, la mise à jour et la réplication de votre infrastructure de manière cohérente et fiable.

Lorsque vous utilisez la AWS::DSQL::Cluster ressource dans vos CloudFormation modèles, vous pouvez provisionner des clusters Aurora DSQL de manière déclarative en même temps que vos autres ressources cloud. Cela permet de s'assurer que votre infrastructure de données se déploie et se gère en même temps que le reste de votre pile d'applications.

Connecteurs pour Aurora DSQL

Aurora DSQL fournit des connecteurs spécialisés qui étendent les pilotes de base de données existants afin de permettre une authentification IAM fluide et une intégration aux AWS services. Ces connecteurs sont conçus pour fonctionner avec les langages de programmation et les frameworks les plus courants tout en maintenant la compatibilité avec les flux de travail PostgreSQL existants.

Des connecteurs supplémentaires sont prévus pour les prochaines versions. Pour obtenir les informations les plus récentes sur la disponibilité des connecteurs, consultez le référentiel d'exemples Aurora DSQL.

Connexion aux clusters Aurora DSQL à l'aide d'un connecteur JDBC

Le connecteur Aurora DSQL pour JDBC est conçu comme un plug-in d'authentification qui étend les fonctionnalités du pilote JDBC PostgreSQL afin de permettre aux applications de s'authentifier auprès d'Aurora DSQL à l'aide des informations d'identification IAM. Le connecteur ne se connecte pas directement à la base de données, mais fournit une authentification IAM fluide en plus du pilote JDBC PostgreSQL sous-jacent.

Le connecteur Aurora DSQL pour JDBC est conçu pour fonctionner avec le pilote JDBC PostgreSQL et assure une intégration parfaite avec les exigences d'authentification IAM d'Aurora DSQL.

Associé au pilote JDBC PostgreSQL, le connecteur Aurora DSQL pour JDBC permet l'authentification basée sur IAM pour Aurora DSQL. Il introduit une intégration approfondie avec les services AWS d'authentification tels que Gestion des identités et des accès AWS(IAM).

À propos du connecteur

Aurora DSQL est un service de base de données SQL distribué qui fournit une disponibilité et une capacité de mise à l'échelle élevées aux applications compatibles avec PostgreSQL. Aurora DSQL nécessite une authentification basée sur IAM avec des jetons limités dans le temps que les pilotes JDBC existants ne prennent pas en charge de manière native.

L'idée principale du connecteur Aurora DSQL pour JDBC est d'ajouter une couche d'authentification au-dessus du pilote JDBC PostgreSQL qui gère la génération de jetons IAM, permettant aux utilisateurs de se connecter à Aurora DSQL sans modifier leurs flux de travail JDBC existants.

Qu'est-ce que l'authentification Aurora DSQL ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l’authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : les jetons d'authentification sont générés à l'aide AWS d'informations d'identification et ont une durée de vie configurable

Le connecteur Aurora DSQL pour JDBC est conçu pour comprendre ces exigences et générer automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Avantages du connecteur Aurora DSQL pour JDBC

Bien qu’Aurora DSQL fournisse une interface compatible avec PostgreSQL, les pilotes PostgreSQL existants ne prennent actuellement pas en charge les exigences d’authentification IAM d’Aurora DSQL. Le connecteur Aurora DSQL pour JDBC permet aux clients de continuer à utiliser leurs flux de travail PostgreSQL existants tout en activant l'authentification IAM via :

- Génération automatique de jetons : les jetons IAM sont générés automatiquement à l'aide d'informations d'identification AWS
- Intégration fluide : compatible avec les modèles de connexion JDBC existants
- AWS Support des informations d'identification : prend en charge différents fournisseurs AWS d'informations d'identification (par défaut, basé sur le profil, etc.)

Utilisation du connecteur SQL Aurora pour JDBC avec regroupement de connexions

Le connecteur Aurora DSQL pour JDBC fonctionne avec des bibliothèques de regroupement de connexions telles que HikariCP. Le connecteur gère la génération de jetons IAM lors de l’établissement de la connexion, ce qui permet aux groupes de connexions de fonctionner normalement.

Fonctions principales

Génération automatique de jetons

Les jetons IAM sont générés automatiquement à l'aide des AWS informations d'identification.

Intégration transparente

Compatible avec les modèles de connexion JDBC existants sans nécessiter de modifications du flux de travail.

AWS Support des informations d'identification

Prend en charge différents fournisseurs AWS d'identifiants (par défaut, basés sur le profil, etc.).

Compatibilité avec le regroupement de connexions

Parfaitement compatible avec les bibliothèques de regroupement de connexions telles que HikariCP.

Conditions préalables

Avant de commencer, assurez-vous de remplir les prérequis suivants :

- Création d'un cluster dans Aurora DSQL.
- Installation du kit de développement Java (JDK). Assurez-vous qu'une version 17 ou supérieure est disponible.
- Configuration des autorisations IAM appropriées pour permettre à votre application de se connecter à Aurora DSQL.
- AWS informations d'identification configurées (via AWS CLI des variables d'environnement ou des rôles IAM).

Utilisation du connecteur SQL Aurora pour JDBC

Pour utiliser le connecteur Aurora DSQL pour JDBC dans votre application Java, procédez comme suit :

1. Ajoutez les dépendances suivantes à votre projet Maven :

```
<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>
```

Pour les projets Gradle, ajoutez cette dépendance :

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

2. Créez une connexion de base à votre cluster Aurora DSQL à l'aide du format du connecteur AWS DSQL PostgreSQL :

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

public class DsqlJdbcConnectorExample {
    public static void main(String[] args) {
        // Using AWS DSQL PostgreSQL Connector prefix
        String jdbcUrl = "jdbc:aws-dsql:postgresql://your-cluster.dsql.us-
east-1.on.aws/postgres?user=admin";

        try (Connection connection = DriverManager.getConnection(jdbcUrl)) {
            // Use the connection
            try (Statement statement = connection.createStatement()) {
                // Create a table
                statement.execute("CREATE TABLE IF NOT EXISTS test_table (id UUID
 PRIMARY KEY DEFAULT gen_random_uuid(), name VARCHAR(100))");

                // Insert data
                statement.execute("INSERT INTO test_table (name) VALUES ('Test
 Name')");

                // Query data
                try (ResultSet resultSet = statement.executeQuery("SELECT * FROM
 test_table")) {
                    while (resultSet.next()) {
                        System.out.println("ID: " + resultSet.getInt("id") + ",
 Name: " + resultSet.getString("name"));
                    }
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

Propriétés de configuration

Le connecteur Aurora DSQL pour JDBC prend en charge les propriétés de connexion suivantes :

user

Détermine l'utilisateur pour la connexion et la méthode de génération de jetons utilisée. Exemple : `admin`

token-duration-secs

Durée de validité du jeton en secondes. Pour plus d'informations sur les limites des jetons, consultez Génération d'un jeton d'authentification dans Amazon Aurora DSQL.

profile

Utilisé pour instancier une génération ProfileCredentialsProvider de jetons avec le nom de profil fourni.

region

AWS région pour les connexions SQL Aurora. Elle est facultative. Lorsqu'elle est fournie, elle remplace la région extraite de l'URL.

database

Le nom de la base de données à laquelle se connecter. La valeur par défaut est `postgres`.

Logging

Activez la journalisation pour résoudre tout problème que vous pourriez rencontrer lors de l'utilisation du connecteur JDBC Aurora DSQL.

Le connecteur utilise le système de journalisation intégré (java.util.logging) de Java. Vous pouvez configurer les niveaux de journalisation en créant un fichier `logging.properties` :

```
# Set root logger level to INFO for clean output
.level = INFO

# Show Aurora DSQL Connector for JDBC FINE logs for detailed debugging
software.amazon.dsql.level = FINE
```

```
# Console handler configuration
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Detailed formatter pattern with timestamp and logger name
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Exemples

Pour des exemples et des cas d'utilisation plus complets, reportez-vous au connecteur Aurora DSQL pour le référentiel JDBC

Connecteur SQL Aurora pour Python

Le connecteur SQL Aurora pour Python intègre l'authentification IAM pour connecter les applications Python aux clusters Amazon Aurora DSQL. En interne, il utilise les bibliothèques clientes psycopg, psycopg2 et asyncpg.

Le connecteur SQL Aurora pour Python est conçu comme un plug-in d'authentification qui étend les fonctionnalités des bibliothèques clientes psycopg, psycopg2 et asyncpg afin de permettre aux applications de s'authentifier auprès d'Amazon Aurora DSQL à l'aide des informations d'identification IAM. Le connecteur ne se connecte pas directement à la base de données mais fournit une authentification IAM fluide en plus des bibliothèques clientes sous-jacentes.

À propos du connecteur

Amazon Aurora DSQL est un service de base de données SQL distribué qui fournit une disponibilité et une évolutivité élevées aux applications compatibles avec PostgreSQL. Aurora DSQL nécessite une authentification basée sur IAM avec des jetons limités dans le temps que les bibliothèques Python existantes ne prennent pas en charge de manière native.

L'idée du connecteur Aurora DSQL pour Python est d'ajouter une couche d'authentification au-dessus des bibliothèques clientes psycopg, psycopg2 et asyncpg qui gère la génération de jetons IAM, permettant aux utilisateurs de se connecter à Aurora DSQL sans modifier leurs flux de travail existants.

Qu'est-ce que l'authentification Aurora DSQL ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l'authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : les jetons d'authentification sont générés à l'aide des informations d'identification AWS et ont une durée de vie configurable

Le connecteur Aurora DSQL pour Python est conçu pour comprendre ces exigences et générer automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Caractéristiques

- Authentification IAM automatique : les jetons IAM sont générés automatiquement à l'aide des informations d'identification AWS
- Construit sur psycopg, psycopg2 et asyncpg : exploite les bibliothèques clientes psycopg, psycopg2 et asyncpg
- Intégration fluide : fonctionne avec les modèles de connexion psycopg, psycopg2 et asyncpg existants sans modification du flux de travail
- Découverte automatique des régions : extrait la région AWS du nom d'hôte du cluster DSQL
- Support des informations d'identification AWS : prend en charge différents fournisseurs d'informations d'identification AWS (par défaut, basé sur le profil, personnalisé)
- Compatibilité avec le regroupement de connexions : fonctionne avec le regroupement de connexions intégré à psycopg, psycopg2 et asyncpg

Guide de démarrage rapide

Exigences

- Python 3.10 ou supérieur
- Accès à un cluster SQL Aurora
- Configuration des autorisations IAM appropriées pour permettre à votre application de se connecter à Aurora DSQL.
- Informations d'identification AWS configurées (via l'interface de ligne de commande AWS, les variables d'environnement ou les rôles IAM)

Installation

```
pip install aurora-dsql-python-connector
```

Installez psycopg ou psycopg2 ou asyncpg séparément

Le programme d'installation d'Aurora DSQL Connector pour Python n'installe pas les bibliothèques sous-jacentes. Ils doivent être installés séparément, par exemple :

```
# Install psycopg and psycopg pool
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
pip install psycopg2-binary
```

```
# Install asyncpg
pip install asyncpg
```

Remarque :

Seule la bibliothèque nécessaire doit être installée. Par conséquent, si le client doit utiliser psycopg, seul psycopg doit être installé. Si le client doit utiliser psycopg2, seul psycopg2 doit être installé. Si le client doit utiliser asyncpg, seul asyncpg doit être installé.

Si le client en a besoin de plusieurs, toutes les bibliothèques nécessaires doivent être installées.

Utilisation de base

psycopie

```
    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)
```

En utilisant uniquement l'hôte

psycopie

```
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")
```

En utilisant uniquement l'ID du cluster

psycopie

```
    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")
```

Remarque :

Dans le scénario « en utilisant uniquement l'identifiant du cluster », la région précédemment définie sur la machine est utilisée, par exemple :

```
aws configure set region us-east-1
```

Si la région n'a pas été définie ou si l'ID de cluster indiqué se trouve dans une autre région, la connexion échouera. Pour que cela fonctionne, indiquez la région en tant que paramètre, comme dans l'exemple ci-dessous :

psycopie

```
    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)
```

Chaîne de connexion

psycopie

```
    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")
```

Configuration avancée

psycopie

```
    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)
```

Options de configuration

Option	Type	Obligatoire	Description
`host`	`string`	Oui	Nom d'hôte ou ID de cluster DSQL
`user`	`string`	Non	Nom d'utilisateur DSQL. Par défaut : admin

Option	Type	Obliga re	Description
`dbname`	`string`	Non	Nom de la base de données. Par défaut : postgres
`region`	`string`	Non	Région AWS (détectée automatiquement à partir du nom d'hôte s'il n'est pas fourni)
`port`	`int`	Non	Par défaut : 5432
`custom_credentials_provider`	`CredentialProvider`	Non	Fournisseur d'informations d'identification AWS personnalisées
`profile`	`string`	Non	Le nom du profil IAM. Par défaut : par défaut.
`token_duration_secs`	`int`	Non	Délai d'expiration du jeton en secondes

Toutes les options de connexion standard des bibliothèques psycopg, psycopg2 et asyncpg sous-jacentes sont également prises en charge, à l'exception des paramètres asyncpg krbsrvname et gsslib qui ne sont pas pris en charge par DSQL.

Utilisation du connecteur SQL Aurora pour Python avec regroupement de connexions

Le connecteur SQL Aurora pour Python fonctionne avec le regroupement de connexions intégré psycopg, psycopg2 et asyncpg. Le connecteur gère la génération de jetons IAM lors de l'établissement de la connexion, ce qui permet aux groupes de connexions de fonctionner normalement.

psycopie

Pour psycopg, le connecteur implémente une classe de connexion nommée DSQLConnection qui peut être transmise directement au psycopg_pool. ConnectionPool constructeur. Pour les opérations asynchrones, il existe également une version asynchrone de la classe nommée 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
)
```

Remarque : Configuration max_lifetime de la connexion

Le paramètre max_lifetime doit être défini sur moins de 3 600 secondes (une heure), car il s'agit de la durée de connexion maximale autorisée par la base de données Aurora DSQL. La définition d'une valeur max_lifetime inférieure permet au pool de connexions de gérer de manière proactive le recyclage des connexions, ce qui est plus efficace que la gestion des erreurs de délai de connexion provenant de la base de données.

psycopg2

Pour psycopg2, le connecteur fournit une classe nommée Aurora DSQLThreaded ConnectionPool qui hérite de psycopg2.pool. ThreadedConnectionPool. La DSQLThreaded ConnectionPool classe Aurora remplace uniquement la méthode interne _connect. Le reste de l'implémentation est fourni par psycopg2.pool. ThreadedConnectionPool inchangé.

```
import aurora_dsql_psycopg2 as dsql

pool = dsql.AuroraDSQLThreadedConnectionPool(
        minconn=2,
        maxconn=8,
        **conn_params,
)
```

asyncpg

Pour asyncpg, le connecteur fournit une fonction create_pool qui renvoie une instance de asyncpg.pool.

```
import asyncio
import os
```

```
    import aurora_dsql_asyncpg as dsql

    pool_params = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'user': "admin",
        "min_size": 2,
        "max_size": 5,
    }

    pool = await dsql.create_pool(**pool_params)
```

Authentification

Le connecteur gère automatiquement l'authentification DSQL en générant des jetons à l'aide du générateur de jetons du client DSQL. Si la région AWS n'est pas fournie, elle sera automatiquement analysée à partir du nom d'hôte fourni.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultez le guide de l'utilisateur.

Administrateur et utilisateurs réguliers

- Les utilisateurs nommés utilisent `"admin"` automatiquement des jetons d'authentification d'administrateur
- Tous les autres utilisateurs utilisent des jetons d'authentification non administrateurs
- Les jetons sont générés dynamiquement pour chaque connexion

Exemples

Pour un exemple de code complet, reportez-vous aux exemples indiqués dans les sections ci-dessous. Pour savoir comment exécuter les exemples, reportez-vous aux exemples de fichiers READMDE.

psycopie

Exemples README

Description	Exemples
Utilisation du connecteur Aurora DSQL pour Python	Exemple de connexion de base

Description	Exemples
pour les connexions de base	
Utilisation du connecteur Aurora DSQL pour Python pour les connexions asynchrones de base	Exemple de connexion asynchrone de base
Utilisation du connecteur Aurora DSQL pour Python avec un pool de connexion s	Exemple de connexion de base avec un pool de connexions
	Exemple de connexions simultanées avec pool de connexions
Utilisation du connecteur SQL Aurora pour Python avec un pool de connexion s asynchrones	Exemple de connexion de base avec un pool de connexions asynchrones

psycopg2

Exemples README

Description	Exemples
Utilisation du connecteur Aurora DSQL pour Python pour les connexions de base	Exemple de connexion de base
Utilisation du connecteur Aurora DSQL pour Python avec un pool de connexion s	Exemple de connexion de base avec un pool de connexions

Description	Exemples
	Exemple de connexions simultanées avec pool de connexions

asyncpg

Exemples README

Description	Exemples
Utilisation du connecteur Aurora DSQL pour Python pour les connexions de base	Exemple de connexion de base
Utilisation du connecteur Aurora DSQL pour Python avec un pool de connexions	Exemple de connexion de base avec un pool de connexions
	Exemple de connexions simultanées avec pool de connexions

Connexion aux clusters SQL Aurora à l'aide d'un connecteur Go

Le connecteur SQL Aurora pour Go intègre l'authentification IAM automatique à pgx. Le connecteur gère la génération de jetons, la configuration SSL et la gestion des connexions afin que vous puissiez vous concentrer sur la logique de votre application.

À propos du connecteur

Aurora DSQL nécessite une authentification basée sur IAM avec des jetons limités dans le temps que les pilotes Go PostgreSQL existants ne prennent pas en charge de manière native. Le connecteur Aurora DSQL pour Go ajoute une couche d'authentification au-dessus du pilote pgx qui gère la génération de jetons IAM, ce qui vous permet de vous connecter à Aurora DSQL sans modifier vos flux de travail pgx existants.

Qu’est-ce que l’authentification Aurora DSQL ?

Dans Aurora DSQL, l’authentification implique :

- Authentification IAM : toutes les connexions utilisent l’authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : le connecteur génère des jetons d'authentification à l'aide AWS d'informations d'identification, et ces jetons ont une durée de vie configurable

Le connecteur Aurora DSQL pour Go est conçu pour comprendre ces exigences et générer automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Avantages du connecteur Aurora DSQL pour Go

Le connecteur SQL Aurora pour Go vous permet de continuer à utiliser vos flux de travail pgx existants tout en activant l'authentification IAM via :

- Génération automatique de jetons : le connecteur génère des jetons IAM automatiquement pour chaque connexion
- Regroupement de connexions : prise en charge intégrée `pgxpool` de la génération automatique de jetons par connexion
- Configuration flexible : Support pour des terminaux complets ou des clusters IDs avec détection automatique des régions
- AWS Support des identifiants : prend en charge les AWS profils et les fournisseurs d'identifiants personnalisés

Fonctions principales

Gestion automatique des jetons

Le connecteur génère automatiquement des jetons IAM pour chaque nouvelle connexion à l'aide d'informations d'identification prérésolues.

Regroupement de connexions

Regroupement des connexions via `pgxpool` génération automatique de jetons par connexion.

Configuration flexible de l'hôte

Prend en charge à la fois les points de terminaison complets du cluster et le cluster IDs avec détection automatique des régions.

Sécurité SSL

Le protocole SSL est toujours activé avec le mode de vérification complète et la négociation directe du protocole TLS.

Conditions préalables

- Go 1.24 ou version ultérieure
- AWS informations d'identification configurées
- Un cluster SQL Aurora

Le connecteur utilise la chaîne d'informations d'identification par défaut du AWS SDK for Go v2, qui résout les informations d'identification dans l'ordre suivant :

1. Variables d'environnement (AWS_ACCESS_KEY_ID, _ACCESS_KEY) AWS_SECRET
2. Fichier d'informations d'identification partagé (~/.aws/credentials)
3. Fichier de configuration partagé (~/.aws/config)
4. Rôle IAM pour Amazon EC2/ECS/Lambda

Installation

Installez le connecteur à l'aide des modules Go :

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql
```

Démarrage rapide

L'exemple suivant montre comment créer un pool de connexions et exécuter une requête :

```
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)
}
```

Options de configuration

Le connecteur prend en charge les options de configuration suivantes :

Champ	Type	Par défaut	Description
Host (Hôte)	chaîne	(obligatoire)	Point de terminaison ou ID de cluster
Région	chaîne	(détecté automatiquement)	AWS région ; obligatoire si l'hôte est un identifiant de cluster
Utilisateur	chaîne	« administrateur »	Utilisateur de la base de donnée
Base de données	chaîne	« postgres »	Nom de la base de données

Champ	Type	Par défaut	Description
Port	int	5432	Port de la base de données
Profil	chaîne	""	AWS nom de profil pour les informations d'identification
TokenDurationSecs	int	900 (15 minutes)	Durée de validité du jeton en secondes (maximum autorisé : 1 semaine, par défaut : 15 min)
MaxConns	int32	0	Nombre maximum de connexions au pool (0 = pgxpool par défaut)
MinConns	int32	0	Nombre minimal de connexions au pool (0 = pgxpool par défaut)
MaxConnLifetime	Heure. Durée	55 minutes	Durée de vie maximale de la connexion

Format de chaîne de connexion

Le connecteur prend en charge les formats de chaîne de connexion PostgreSQL et DSQL :

```
postgres://[user@]host[:port]/[database][?param=value&...]
dsql://[user@]host[:port]/[database][?param=value&...]
```

Paramètres de requête pris en charge :

- `region`- AWS région
- `profile`- nom du AWS profil
- `tokenDurationSecs`- Durée de validité du jeton en secondes

Exemples :

```
// 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")
```

Utilisation avancée

Configuration de l'hôte

Le connecteur prend en charge deux formats d'hôte :

Point de terminaison complet (région détectée automatiquement) :

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
})
```

ID du cluster (région requise) :

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host:   "your-cluster-id",
    Region: "us-east-1",
})
```

Réglage de la configuration du pool

Configurez le pool de connexions pour votre charge de travail :

```
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,
})
```

Utilisation d'une connexion unique

Pour les scripts simples ou lorsque le regroupement de connexions n'est pas nécessaire :

```
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")
```

Utilisation de AWS profils

Spécifiez un AWS profil pour les informations d'identification :

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:    "your-cluster.dsql.us-east-1.on.aws",
    Profile: "production",
})
```

Réessayer OCC

Aurora DSQL utilise un contrôle de simultanéité optimiste (OCC). Lorsque deux transactions modifient les mêmes données, la première à valider gagne et la seconde reçoit une erreur OCC.

Le `occretry` package fournit des aides pour une nouvelle tentative automatique avec un recul et une instabilité exponentiels. Installez-le avec :

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/occretry
```

Utilisation `WithRetry` pour les écritures transactionnelles :

```
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
})
```

Pour les instructions DDL ou uniques, utilisez `ExecWithRetry` :

```
err := occretry.ExecWithRetry(ctx, pool, occretry.DefaultConfig(),
    "CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY, name TEXT)")
```

> **Important**
>
> `WithRetrygèreBEGIN/COMMIT/ROLLBACK`en interne. Votre rappel reçoit une transaction et ne doit contenir que des opérations de base de données et vous pouvez réessayer en toute sécurité.

Exemples

Pour des exemples et des cas d'utilisation plus complets, reportez-vous aux exemples d'Aurora DSQL Connector for Go.

Exemple	Description
exemple_préféré	Recommandé : pool de connexions avec requêtes simultanées
transaction	Gestion des transactions avec BEGIN/COMMIT/ROLLBACK
occ_réessayer	Gérer les conflits OCC avec un recul exponentiel
chaîne_connexion	Utilisation de chaînes de connexion pour la configuration

Connecteurs SQL Aurora pour Node.js

Le connecteur Aurora DSQL pour node-postgres et le connecteur Aurora DSQL pour Postgres.js sont des plug-ins d'authentification qui étendent les fonctionnalités des clients node-postgres et Postgres.js afin de permettre aux applications de s'authentifier auprès d'Aurora DSQL à l'aide des informations d'identification IAM.

Connecteur SQL Aurora pour node-postgres

Le connecteur Aurora DSQL pour node-postgres est un connecteur Node.js basé sur node-postgres qui intègre l'authentification IAM pour connecter des applications aux clusters Amazon Aurora DSQL. JavaScript/TypeScript

Le connecteur Aurora DSQL est conçu comme un plug-in d'authentification qui étend les fonctionnalités du client et du pool de nœuds postgres afin de permettre aux applications de s'authentifier auprès d'Amazon Aurora DSQL à l'aide des informations d'identification IAM.

À propos du connecteur

Amazon Aurora DSQL est une base de données distribuée native dans le cloud compatible avec PostgreSQL. Bien qu'il nécessite une authentification IAM et des jetons limités dans le temps, les pilotes de base de données Node.js traditionnels ne disposent pas de cette prise en charge intégrée.

Le connecteur SQL Aurora pour node-postgres comble cette lacune en implémentant un intergiciel d'authentification qui fonctionne parfaitement avec node-postgres. Cette approche permet aux développeurs de conserver leur code node-postgres existant tout en obtenant un accès sécurisé basé sur l'IAM aux clusters Aurora DSQL grâce à la gestion automatisée des jetons.

Qu'est-ce que l'authentification Aurora DSQL ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l'authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : les jetons d'authentification sont générés à l'aide des informations d'identification AWS et ont une durée de vie configurable

Le connecteur Aurora DSQL pour node-postgres est conçu pour comprendre ces exigences et générer automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Caractéristiques

- Authentification IAM automatique : gère la génération et l'actualisation des jetons DSQL
- Construit sur node-postgres : exploite le célèbre client PostgreSQL pour Node.js
- Intégration fluide - Fonctionne avec les modèles de connexion node-postgres existants
- Découverte automatique des régions : extrait la région AWS du nom d'hôte du cluster DSQL
- TypeScript Support complet - Assure une sécurité totale du type

- Support des informations d'identification AWS : prend en charge différents fournisseurs d'informations d'identification AWS (par défaut, basé sur le profil, personnalisé)
- Compatibilité avec le regroupement de connexions : fonctionne parfaitement grâce au regroupement de connexions intégré

Exemple d'application

Un exemple d'application est inclus, par exemple, qui montre comment utiliser le connecteur Aurora DSQL pour node-postgres. Pour exécuter l'exemple inclus, veuillez vous référer à l'exemple README.

Guide de démarrage rapide

Exigences

- Node.js 20+
- Accès à un cluster SQL Aurora
- Configuration des autorisations IAM appropriées pour permettre à votre application de se connecter à Aurora DSQL.
- Informations d'identification AWS configurées (via l'interface de ligne de commande AWS, les variables d'environnement ou les rôles IAM)

Installation

```
npm install @aws/aurora-dsql-node-postgres-connector
```

Dépendances entre pairs

```
npm install @aws-sdk/credential-providers @aws-sdk/dsql-signer pg tsx
npm install --save-dev @types/pg
```

Usage

Connexion client

```
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "<CLUSTER_ENDPOINT>",
```

```
  user: "admin",
});
await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Connexion à la piscine

```
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()");
```

Utilisation avancée

```
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();
```

Options de configuration

Option	Type	Obli re	Description
`host`	`string`	Oui	Nom d'hôte du cluster SQL

Option	Type	Obli re	Description
`username`	`string`	Oui	Nom d'utilisateur DSQL
`database`	`string`	Nor	Nom de la base de données
`region`	`string`	Nor	Région AWS (détectée automatiquement à partir du nom d'hôte s'il n'est pas fourni)
`port`	`number`	Nor	Par défaut : 5432
`customCredentialsProvider`	`AwsCredentialIdentity / AwsCredentialIdentityProvider`	Nor	Fournisseur d'informations d'identification AWS personnalisées
`profile`	`string`	Nor	Le nom du profil IAM. Par défaut, c'est « par défaut »
`tokenDurationSecs`	`number`	Nor	Délai d'expiration du jeton en secondes

Tous les autres paramètres du Client/Pool sont pris en charge.

Authentification

Le connecteur gère automatiquement l'authentification DSQL en générant des jetons à l'aide du générateur de jetons du client DSQL. Si la région AWS n'est pas fournie, elle sera automatiquement analysée à partir du nom d'hôte fourni.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultez le guide de l'utilisateur.

Administrateur et utilisateurs réguliers

- Les utilisateurs nommés « admin » utilisent automatiquement des jetons d'authentification d'administrateur
- Tous les autres utilisateurs utilisent des jetons d'authentification ordinaires
- Les jetons sont générés dynamiquement pour chaque connexion

Connecteur SQL Aurora pour Postgres.js

Le connecteur Aurora DSQL pour Postgres.js est un connecteur Node.js basé sur Postgres.js qui intègre l'authentification IAM pour connecter des JavaScript applications aux clusters Amazon Aurora DSQL.

Le connecteur Aurora DSQL pour Postgres.js est conçu comme un plug-in d'authentification qui étend les fonctionnalités du client Postgres.js afin de permettre aux applications de s'authentifier auprès d'Amazon Aurora DSQL à l'aide des informations d'identification IAM. Le connecteur ne se connecte pas directement à la base de données, mais fournit une authentification IAM fluide en plus du pilote Postgres.js sous-jacent.

À propos du connecteur

Amazon Aurora DSQL est un service de base de données SQL distribué qui fournit une disponibilité et une évolutivité élevées aux applications compatibles avec PostgreSQL. Aurora DSQL nécessite une authentification basée sur IAM avec des jetons limités dans le temps que les pilotes Node.js existants ne prennent pas en charge de manière native.

L'idée du connecteur Aurora DSQL pour Postgres.js est d'ajouter une couche d'authentification au-dessus du client Postgres.js qui gère la génération de jetons IAM, permettant aux utilisateurs de se connecter à Aurora DSQL sans modifier leurs flux de travail Postgres.js existants.

Le connecteur Aurora DSQL pour Postgres.js fonctionne avec la plupart des versions de Postgres.js. Les utilisateurs fournissent leur propre version en installant directement Postgres.js.

Qu'est-ce que l'authentification Aurora DSQL ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l'authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : les jetons d'authentification sont générés à l'aide des informations d'identification AWS et ont une durée de vie configurable

Le connecteur Aurora DSQL pour Postgres.js est conçu pour comprendre ces exigences et générer automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Caractéristiques

- Authentification IAM automatique : gère la génération et l'actualisation des jetons DSQL

- Construit sur Postgres.js : exploite le client rapide PostgreSQL pour Node.js
- Intégration parfaite - Fonctionne avec les modèles de connexion Postgres.js existants
- Découverte automatique des régions : extrait la région AWS du nom d'hôte du cluster DSQL
- TypeScript Support complet - Assure une sécurité totale du type
- Support des informations d'identification AWS : prend en charge différents fournisseurs d'informations d'identification AWS (par défaut, basé sur le profil, personnalisé)
- Compatibilité avec le regroupement de connexions - Fonctionne parfaitement avec le regroupement de connexions intégré à Postgres.js

Guide de démarrage rapide

Exigences

- Node.js 20+
- Accès à un cluster SQL Aurora
- Configuration des autorisations IAM appropriées pour permettre à votre application de se connecter à Aurora DSQL.
- Informations d'identification AWS configurées (via l'interface de ligne de commande AWS, les variables d'environnement ou les rôles IAM)

Installation

```
npm install @aws/aurora-dsql-postgresjs-connector
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Utilisation de base

```
import { auroraDSQLPostgres } from '@aws/aurora-dsql-postgresjs-connector';

const sql = auroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  username: 'admin',

});

// Execute queries
const result = await sql`SELECT current_timestamp`;
console.log(result);
```

```
// Clean up
await sql.end();
```

Utilisation de l'ID du cluster au lieu de l'hôte

```
const sql = auroraDSQLPostgres({
  host: 'your-cluster-id',
  region: 'us-east-1',
  username: 'admin',

});
```

Chaîne de connexion

```
const sql = AuroraDSQLPostgres(
  'postgres://admin@your-cluster.dsql.us-east-1.on.aws'
);

const result = await sql`SELECT current_timestamp`;
```

Configuration avancée

```
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
});
```

Options de configuration

Option	Type	Obliga re	Description
`host`	`string`	Oui	Nom d'hôte ou ID de cluster DSQL
`database`	`string?`	Non	Nom de la base de données
`username`	`string?`	Non	Nom d'utilisateur de la base de données (utilise admin s'il n'est pas fourni)
`region`	`string?`	Non	Région AWS (détectée automatiquement à partir du nom d'hôte s'il n'est pas fourni)
`customCredentialsProvider`	`AwsCredentialIdentityProvider?`	Non	Fournisseur d'informations d'identification AWS personnalisées
`tokenDurationSecs`	`number?`	Non	Délai d'expiration du jeton en secondes

Toutes les options standard de Postgres.js sont également prises en charge.

Authentification

Le connecteur gère automatiquement l'authentification DSQL en générant des jetons à l'aide du générateur de jetons du client DSQL. Si la région AWS n'est pas fournie, elle sera automatiquement analysée à partir du nom d'hôte fourni.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultez le guide de l'utilisateur.

Administrateur et utilisateurs réguliers

- Les utilisateurs nommés « admin » utilisent automatiquement des jetons d'authentification d'administrateur
- Tous les autres utilisateurs utilisent des jetons d'authentification ordinaires
- Les jetons sont générés dynamiquement pour chaque connexion

Utilisation de l'échantillon

JavaScript des exemples utilisant le connecteur Aurora DSQL pour Postgres.js sont disponibles sur GitHub. Pour obtenir des instructions sur la façon d'exécuter les exemples, reportez-vous au répertoire des exemples.

Description	Exemple
Regroupement de connexions avec requêtes simultanées, y compris la création de tables, les insertions et les lectures entre plusieurs travailleurs	Exemple de pool de connexions (préféré)
Opérations CRUD (création de table, insertion, sélection, suppression) sans regroupement de connexions	Exemple sans pool de connexions

Connexion aux clusters SQL Aurora à l'aide d'un connecteur Ruby

Le connecteur Aurora DSQL pour Ruby est un connecteur Ruby basé sur pg qui intègre l'authentification IAM pour connecter les applications Ruby aux clusters Amazon Aurora DSQL.

Le connecteur gère la génération de jetons, la configuration SSL et le regroupement des connexions afin que vous puissiez vous concentrer sur la logique de votre application.

À propos du connecteur

Amazon Aurora DSQL nécessite une authentification IAM avec des jetons à durée limitée que les pilotes Ruby PostgreSQL existants ne prennent pas en charge de manière native. Le connecteur Aurora DSQL pour Ruby ajoute une couche d'authentification au-dessus de la gem pg qui gère la génération de jetons IAM, vous permettant de vous connecter à Aurora DSQL sans modifier vos flux de travail pg existants.

Qu'est-ce que l'authentification SQL Aurora ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l'authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : le connecteur génère des jetons d'authentification à l'aide AWS d'informations d'identification, et ces jetons ont une durée de vie configurable

Le connecteur Aurora DSQL pour Ruby comprend ces exigences et génère automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Caractéristiques

- Authentification IAM automatique : gère la génération et l'actualisation des jetons Aurora DSQL
- Construit sur pg - Enveloppe la célèbre gemme PostgreSQL pour Ruby
- Intégration parfaite - Fonctionne avec les flux de travail pg gem existants
- Regroupement de connexions - Support intégré via le `connection_pool` gem avec max_lifetime enforcement
- Détection automatique des régions : extrait AWS la région du nom d'hôte du cluster Aurora DSQL
- AWS support des identifiants : prend en charge les AWS profils et les fournisseurs d'identifiants personnalisés
- Nouvelle tentative OCC - Optez pour une nouvelle tentative de contrôle de simultanéité optimiste avec un recul exponentiel

Exemple d'application

Pour un exemple complet, consultez l'exemple d'application sur GitHub.

Guide de démarrage rapide

Exigences

- Ruby 3.1 ou version ultérieure
- Accès à un cluster SQL Aurora
- AWS informations d'identification configurées (via la AWS CLI, les variables d'environnement ou les rôles IAM)

Installation

Ajoutez à votre Gemfile :

```
gem "aurora-dsql-ruby-pg"
```

Ou installez directement :

```
gem install aurora-dsql-ruby-pg
```

Usage

Connexion à la piscine

```
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
```

Connexion simple

Pour les scripts simples ou lorsque le regroupement de connexions n'est pas nécessaire :

```
conn = AuroraDsql::Pg.connect(host: "your-cluster.dsql.us-east-1.on.aws")
conn.exec("SELECT 1")
conn.close
```

Utilisation avancée

Configuration de l'hôte

Le connecteur prend en charge à la fois les points de terminaison complets du cluster (région détectée automatiquement) et le cluster IDs (région requise) :

```
# 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 profils

Spécifiez un AWS profil pour les informations d'identification :

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  profile: "production"
)
```

Format de chaîne de connexion

Le connecteur prend en charge les formats de chaîne de connexion PostgreSQL :

```
postgres://[user@]host[:port]/[database][?param=value&...]
postgresql://[user@]host[:port]/[database][?param=value&...]
```

Paramètres de requête pris en charge :`region`,`profile`,`tokenDurationSecs`.

```
# Full endpoint with profile
```

```
pool = AuroraDsql::Pg.create_pool(
  "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?profile=dev"
)
```

Réessayer OCC

Aurora DSQL utilise un contrôle de simultanéité optimiste (OCC). Lorsque deux transactions modifient les mêmes données, la première à valider gagne et la seconde reçoit une erreur OCC.

Une nouvelle tentative d'OCC est facultative. Configurez `occ_max_retries` lors de la création du pool pour activer la nouvelle tentative automatique avec un recul et une instabilité exponentiels : `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`hn'intègre PAS automatiquement votre bloc dans une transaction. Vous devez `conn.transaction` vous appeler pour les opérations d'écriture. En cas de conflit OCC, le connecteur réexécute l'intégralité du bloc. Il doit donc contenir uniquement les opérations de base de données et pouvoir réessayer en toute sécurité.

Pour éviter de réessayer un appel individuel, passez `retry_occ: false` :

```
pool.with(retry_occ: false) do |conn|
  conn.exec("SELECT 1")
end
```

Options de configuration

Champ	Type	Par défaut	Description
hôte	String	(obligatoire)	Point de terminaison ou ID de cluster
Région	String	(détecté automatiquement)	AWS région ; obligatoire si l'hôte est un ID de cluster
user	String	« administrateur »	Utilisateur de la base de donnée
database	String	« postgres »	Nom de la base de données
port	Entier	5432	Port de la base de données
profile	String	néant	AWS nom de profil pour les informations d'identification
durée du jeton	Entier	900 (15 minutes)	Durée de validité du jeton en secondes (maximum autorisé : 1 semaine, par défaut : 15 min)
credentials_provider	Aws : :Informations d'identification	néant	Fournisseur d'informations d'identification personnalisées
durée de vie maximale	Entier	300 (55 minutes)	Durée de vie maximale de la connexion en secondes
application_name	String	néant	Préfixe ORM pour application_name
bûcheron	Logger	néant	Enregistreur pour les avertissements relatifs aux nouvelles tentatives d'OCC

Champ	Type	Par défaut	Description
occ_max_r éessais	Entier	nil (désactivé)	Max OCC réessaie `pool.with` ; permet de réessayer une fois défini

`create_pool`accepte également un `pool:` mot clé avec un hachage d'options auquel vous passez directement. `ConnectionPool.new` Si vous omettez`pool:`, le connecteur est défini par défaut sur. `{size: 5, timeout: 5}` Les clés que vous fournissez remplacent uniquement ces valeurs par défaut spécifiques.

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  pool: { size: 10, timeout: 10 }
)
```

Authentification

Le connecteur gère automatiquement l'authentification Aurora DSQL en générant des jetons à l'aide d' AWS informations d'identification. Si vous ne fournissez pas la AWS région, le connecteur l'analyse à partir du nom d'hôte.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultez[Authentification et autorisation pour Aurora DSQL](#).

Administrateur et utilisateurs réguliers

- Les utilisateurs nommés « admin » utilisent automatiquement des jetons d'authentification d'administrateur
- Tous les autres utilisateurs utilisent des jetons d'authentification ordinaires
- Le connecteur génère des jetons de manière dynamique pour chaque connexion

Connexion aux clusters SQL Aurora à l'aide d'un connecteur .NET

Le [connecteur Amazon Aurora DSQL pour .NET](#) est un connecteur .NET basé sur [Npgsql](#) qui intègre l'authentification IAM pour connecter les applications .NET aux clusters Amazon Aurora DSQL.

Le connecteur gère la génération de jetons, la configuration SSL et le regroupement des connexions afin que vous puissiez vous concentrer sur la logique de votre application.

À propos du connecteur

Amazon Aurora DSQL nécessite une authentification IAM avec des jetons à durée limitée que les pilotes .NET PostgreSQL existants ne prennent pas en charge de manière native. Le connecteur Aurora DSQL pour .NET ajoute une couche d'authentification au-dessus de Npgsql qui gère la génération de jetons IAM, vous permettant de vous connecter à Aurora DSQL sans modifier vos flux de travail Npgsql existants.

Qu'est-ce que l'authentification SQL Aurora ?

Dans Aurora DSQL, l'authentification implique :

- Authentification IAM : toutes les connexions utilisent l'authentification basée sur IAM avec des jetons à durée limitée
- Génération de jetons : le connecteur génère des jetons d'authentification à l'aide AWS d'informations d'identification, et ces jetons ont une durée de vie configurable

Le connecteur Aurora DSQL pour .NET comprend ces exigences et génère automatiquement des jetons d'authentification IAM lors de l'établissement de connexions.

Caractéristiques

- Authentification IAM automatique : gère la génération et l'actualisation des jetons Aurora DSQL
- Construit sur Npgsql : intègre le célèbre pilote PostgreSQL pour .NET
- Intégration fluide - Fonctionne avec les flux de travail Npgsql existants
- Regroupement de connexions - Support intégré via `NpgsqlDataSource` avec application de la durée de vie maximale
- Détection automatique des régions : extrait AWS la région du nom d'hôte du cluster Aurora DSQL
- AWS support des identifiants - Supporte les AWS profils et les fournisseurs d'identifiants personnalisés
- Nouvelle tentative OCC - Optez pour une nouvelle tentative de contrôle de simultanéité optimiste avec un recul exponentiel
- Application du protocole SSL : utilise toujours le protocole SSL avec `verify-full` mode et négociation directe du protocole TLS

Exemple d'application

Pour un exemple complet, consultez l'exemple d'application sur GitHub.

Guide de démarrage rapide

Exigences

- .NET 8.0 ou version ultérieure
- Accès à un cluster SQL Aurora
- AWS informations d'identification configurées (via la AWS CLI, les variables d'environnement ou les rôles IAM)

Installation

Ajoutez le package à votre projet :

```
dotnet add package Amazon.AuroraDsql.Npgsql
```

Usage

Connexion à la piscine

```
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();
});
```

Connexion simple

Pour les scripts simples ou lorsque vous n'avez pas besoin d'un regroupement de connexions :

```
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();
```

Réessayer OCC

Aurora DSQL utilise un contrôle de simultanéité optimiste (OCC). Lorsque deux transactions modifient les mêmes données, la première à valider gagne et la seconde reçoit une erreur OCC.

Une nouvelle tentative d'OCC est facultative. Définissez `OccMaxRetries` la configuration pour activer la nouvelle tentative automatique avec un recul et une instabilité exponentiels. Utilisation `WithTransactionRetryAsync` pour les écritures transactionnelles :

```
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();
});
```

Pour les instructions DDL ou uniques, utilisez `ExecWithRetryAsync` :

```
await ds.ExecWithRetryAsync("CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY,
 name TEXT)");
```

> **Important**
>
> `WithTransactionRetryAsyncgèreBEGIN/COMMIT/ROLLBACKen` interne et ouvre une nouvelle connexion à chaque tentative. Votre rappel ne doit contenir que des opérations de base de données et vous pouvez réessayer en toute sécurité.

Options de configuration

Le connecteur accepte `postgres://` et `postgresql://` connecte également des chaînes avec `region` des paramètres de `profile` requête.

Champ	Type	Par défaut	Description
`Host`	`string`	(obligatoire)	Point de terminaison du cluster ou ID de cluster à 26 caractères
`Region`	`string?`	(détecté automatiq uement)	AWS région ; obligatoire s'il s'`Host`agit d'un ID de cluster
`User`	`string`	`"admin"`	Utilisateur de la base de donnée
`Database`	`string`	`"postgres"`	Nom de la base de données
`Port`	`int`	`5432`	Port de la base de données
`Profile`	`string?`	`null`	AWS nom de profil pour les informati ons d'identification

Champ	Type	Par défaut	Description
`CustomCredentialsProvider`	`AWSCredentials?`	`null`	Fournisseur AWS d'identifiants personnalisés
`TokenDurationSecs`	`int?`	`null`(SDK par défaut, 900s)	Durée de validité du jeton en secondes
`OccMaxRetries`	`int?`	`null`(handicapé)	Nombre maximal de tentatives OCC par défaut pour les méthodes de nouvelle tentative sur la source de données
`OrmPrefix`	`string?`	`null`	préfixe ORM préfixé à `application_name`
`LoggerFactory`	`ILoggerFactory?`	`null`	Logger Factory pour les nouvelles tentatives, les avertissements et les diagnostics
`ConfigureConnectionString`	`Action<NpgsqlConnectionStringBuilder>?`	`null`	Rappel pour annuler les paramètres du pool ou définir des propriétés de chaîne de connexion Npgsql supplémentaires. Les protocoles SSL `Enlist` sont des invariants de sécurité et ne peuvent pas être remplacés.

Authentification

Le connecteur gère automatiquement l'authentification Aurora DSQL en générant des jetons à l'aide d' AWS informations d'identification. Si vous ne fournissez pas la AWS région, le connecteur l'analyse à partir du nom d'hôte.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultezAuthentification et autorisation pour Aurora DSQL.

Administrateur et utilisateurs réguliers

- Les utilisateurs nommés « admin » utilisent automatiquement des jetons d'authentification d'administrateur
- Tous les autres utilisateurs utilisent des jetons d'authentification ordinaires
- Le connecteur génère des jetons de manière dynamique pour chaque connexion.

Accès à Aurora DSQL avec des clients compatibles avec PostgreSQL

Aurora DSQL utilise le protocole filaire PostgreSQL. Vous pouvez vous connecter à PostgreSQL à l'aide de divers outils et clients, AWS CloudShell tels que DBeaver psql et. DataGrip Le tableau suivant résume la façon dont Aurora DSQL mappe les paramètres de connexion PostgreSQL courants :

PostgreSQL	Aurora DSQL	Remarques
Rôle (également connu sous le nom d'utilisateur ou de groupe)	Rôle de base de données	Aurora DSQL crée pour vous un rôle nommé `admin`. Lorsque vous créez des rôles de base de données personnalisés, vous devez utiliser le rôle `admin` pour les associer à des rôles IAM afin de vous authentifier lors de la connexion à votre cluster. Pour de plus amples informations, veuillez consulter Utilisation des rôles de base de données et de l'authentification IAM.
Hôte (également appelé nom d'hôte ou spécification d'hôte)	Point de terminaison de cluster	Les clusters à une seule région Aurora DSQL fournissent un point de terminaison géré unique et redirigent automatiquement le trafic en cas d'indisponibilité dans la région.
Port	N/A : utiliser la valeur par défaut `5432`	Il s'agit de la valeur par défaut de PostgreSQL.

PostgreSQL	Aurora DSQL	Remarques
Base de données (dbname)	utiliser `postgres`	Aurora DSQL crée cette base de données pour vous lorsque vous créez le cluster.
Mode SSL	Le protocole SSL est toujours activé côté serveur	Dans Aurora DSQL, Aurora DSQL prend en charge le mode SSL `require`. Les connexions sans SSL sont rejetées par Aurora DSQL.
Mot de passe	Jeton d'authentification	Aurora DSQL nécessite des jetons d'authentification temporaires plutôt que des mots de passe de longue durée. Pour en savoir plus, veuillez consulter la section Création d'un jeton d'authentification dans Amazon Aurora DSQL.

Lors de la connexion, Aurora DSQL nécessite un jeton d'authentification IAM signé à la place d'un mot de passe traditionnel. Ces jetons temporaires sont générés à l'aide de AWS la version 4 de Signature et ne sont utilisés que lors de l'établissement de la connexion. Une fois connectée, la session reste active jusqu'à ce qu'elle se termine ou que le client se déconnecte.

Si vous tentez d'ouvrir une nouvelle session avec un jeton expiré, la demande de connexion échoue et un nouveau jeton doit être généré. Pour de plus amples informations, veuillez consulter Création d'un jeton d'authentification dans Amazon Aurora DSQL.

Accédez à Aurora DSQL à l'aide de clients SQL

Aurora DSQL prend en charge plusieurs clients compatibles avec PostgreSQL pour la connexion à votre cluster. Les sections suivantes décrivent comment se connecter à l'aide de AWS CloudShell PostgreSQL ou de votre ligne de commande locale, ainsi que d'outils basés sur une interface graphique tels que et. DBeaver JetBrains DataGrip Chaque client a besoin d'un jeton d'authentification valide, comme décrit dans la section précédente.

Rubriques

- DBeaver À utiliser pour accéder à Aurora DSQL
- JetBrains DataGrip À utiliser pour accéder à Aurora DSQL

- Utiliser le terminal interactif PostgreSQL (psql) pour accéder à Aurora DSQL
- Utilisez le pilote Aurora DSQL pour SQLTools
- Résolution des problèmes

DBeaver À utiliser pour accéder à Aurora DSQL

DBeaver est un client SQL universel qui peut être utilisé pour gérer n'importe quelle base de données dotée d'un pilote JDBC. Il est largement utilisé par les développeurs et les administrateurs de bases de données en raison de ses fonctionnalités robustes de visualisation, d'édition et de gestion des données. Grâce DBeaver aux options de connectivité au cloud, vous pouvez vous connecter DBeaver nativement à Aurora DSQL.

DBeaver Pro

DBeaver Les produits PRO offrent une intégration native avec Aurora DSQL à partir de la version 25.3. Suivez les instructions de la DBeaver documentation pour vous connecter à votre cluster Aurora DSQL.

DBeaver Édition communautaire

DBeaver Community Edition est la version gratuite et open source. Consultez la page de téléchargement pour obtenir les instructions d'installation. Pour vous connecter à DSQL depuis DBeaver Community Edition, vous devez installer le plug-in Aurora DSQL pour. DBeaver

Le plug-in Aurora DSQL pour DBeaver est basé sur le connecteur Aurora DSQL pour JDBC et permet l'authentification IAM sur les clusters Aurora DSQL. Il est facilement installé via l' DBeaver interface utilisateur et élimine le besoin d'écrire le code de génération du jeton ou de fournir manuellement un jeton IAM valide, simplifiant ainsi l'authentification tout en éliminant les risques de sécurité associés aux mots de passe traditionnels générés par les utilisateurs.

Caractéristiques

- Support de l'authentification IAM : connectez-vous aux clusters SQL Aurora à l'aide des informations d'identification AWS IAM pour une authentification sécurisée et sans mot de passe
- Gestion automatique des pilotes : installe et configure en toute simplicité le connecteur Aurora DSQL pour JDBC
- Options de connexion flexibles : choisissez entre une configuration de connexion basée sur l'hôte ou basée sur une URL JDBC

Plug-in SQL Aurora pour l'installation DBeaver

1. Une DBeaver fois ouvert, allez dans le menu déroulant Aide → Installer un nouveau logiciel
2. Cliquez sur Ajouter pour ajouter un nouveau référentiel
3. Entrez :
   - Nom: `Aurora DSQL Plugin`
   - Lieu : `https://awslabs.github.io/aurora-dsql-dbeaver-plugin/update-site/`
4. Vérifiez le connecteur SQL Aurora pour JDBC
5. Cliquez sur Suivant, acceptez la licence et terminez l'installation
6. Redémarrer DBeaver lorsque vous y êtes invité

Création d'une connexion SQL Aurora

1. Cliquez sur le bouton Nouvelle connexion à la base de données
2. Sélectionnez Aurora DSQL
3. Sous Serveur, sélectionnez l'une des options suivantes pour le paramètre Connect by
   - Host
     - pour activer la saisie de texte dans l'interface utilisateur pour les champs suivants :
       - Point de terminaison : point de terminaison du cluster SQL
       - Nom d'utilisateur : nom d'utilisateur DSQL (par exemple admin)
       - Profil AWS : par exemple, par défaut : profil standard utilisé lorsqu'aucun profil spécifique n'est spécifié
       - Région AWS (facultatif) : doit correspondre à la région dans laquelle se trouve votre cluster SQL, sinon l'authentification échouera
   - URL
     - URL JDBC au format suivant :

       ```
       jdbc:aws-dsql:postgresql://{cluster_endpoint}/{database}?
       user=admin&profile=default&region=us-east-1
       ```

     - Remarque : Dans ce mode, seule la saisie d'URL est activée. Pour ajouter des paramètres à la chaîne de connexion JDBC, utilisez le format des paramètres de requête URL commençant par ? comme premier paramètre et ajoutez un & pour les paramètres suivants.
4. Cliquez sur Tester la connexion pour vérifier que la connexion Aurora DSQL fonctionne.
5. Cliquez sur Terminer

Résolution des problèmes

Problème avec le Windows Trust Store

Les utilisateurs de Windows peuvent rencontrer des problèmes lors du téléchargement du pilote Aurora DSQL Connector pour JDBC depuis Maven Central.

Cause : Windows Trust Store peut ne pas inclure les certificats requis pour accéder au référentiel Maven Central.

Solution :

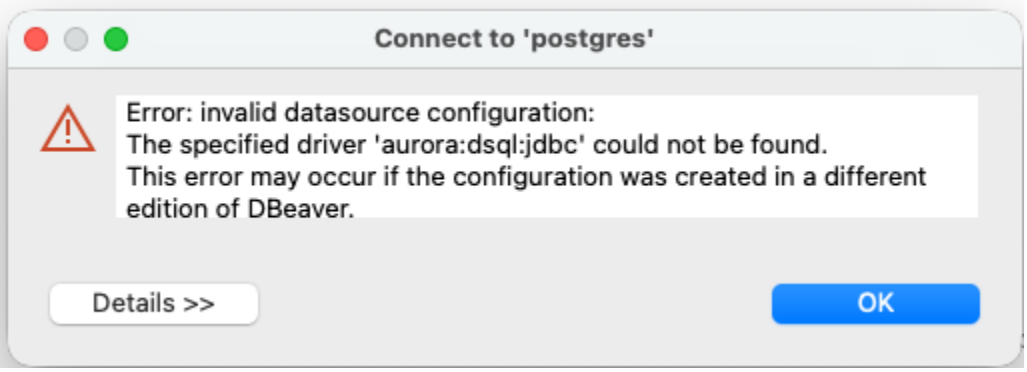
1. Exécuter DBeaver en tant qu' « administrateur »
2. Décochez ce paramètre : Windows **>** Préférences **>** Connexions **>** « Utiliser Windows Trust Store »

Erreur de pilote manquant

Si une icône de pilote est manquante ou si des erreurs de connexion s'affichent, il est possible que le plug-in communautaire Aurora DSQL ne soit pas installé dans votre DBeaver version actuelle. Voici quelques exemples d'erreurs et comment les corriger :

- Création d'une nouvelle connexion avec le pilote manquant :

- Tentative de connexion sans le pilote :



Cause : Lorsque plusieurs DBeaver versions sont installées, les paramètres de connexion sont partagés mais les pilotes sont installés par application.

Solution : Réinstallez Aurora DSQL (plug-in communautaire) en suivant les étapes d'installation ci-dessus.

> **Important**
>
> Les fonctionnalités administratives fournies par DBeaver les bases de données PostgreSQL (telles que Session Manager et Lock Manager) ne s'appliquent pas aux bases de données Aurora DSQL en raison de leur architecture unique. Bien qu'accessibles, ces écrans ne fournissent pas d'informations fiables sur l'état ou l'état de la base de données.

JetBrains DataGrip À utiliser pour accéder à Aurora DSQL

JetBrains DataGrip est un IDE multiplateforme permettant de travailler avec le SQL et les bases de données, y compris PostgreSQL. DataGrip inclut une interface graphique robuste avec un éditeur SQL intelligent. Pour le télécharger DataGrip, rendez-vous sur la page de téléchargement du JetBrainssite Web.

Pour configurer une nouvelle connexion Aurora DSQL dans JetBrains DataGrip

1. Choisissez Nouvelle source de données, puis PostgreSQL.
2. Dans l'onglet Sources de données/Généralités, entrez les informations suivantes :

   - Hôte : utilisez le point de terminaison de votre cluster.

     Port : Aurora DSQL utilise la valeur par défaut de PostgreSQL, `5432`

Base de données : Aurora DSQL utilise la valeur par défaut de PostgreSQL, `postgres`

Authentification : choisissez `User & Password` .

Nom d'utilisateur : entrez `admin`.

Mot de passe : générez un jeton et collez-le dans ce champ.

URL : ne modifiez pas ce champ. Il sera rempli automatiquement en fonction des autres champs.

3. Mot de passe : pour définir cette valeur, générez un jeton d'authentification. Copiez le résultat du générateur de jetons et collez-le dans le champ du mot de passe.

> **Note**
>
> Vous devez définir le mode SSL dans les connexions client. Aurora DSQL prend en charge `PGSSLMODE=require and PGSSLMODE=verify-full`. Aurora DSQL impose la communication SSL côté serveur et rejette les connexions non-SSL. Pour `verify-full` cette option, vous devez installer les certificats SSL localement. Pour plus d'informations, consultez la section Certificats SSL/TLS.

4. Vous devez être connecté à votre cluster et pouvez commencer à exécuter des instructions SQL :

> **Important**
>
> Certaines vues fournies par DataGrip les bases de données PostgreSQL (telles que les sessions) ne s'appliquent pas aux bases de données Aurora DSQL en raison de leur architecture unique. Bien qu'accessibles, ces écrans ne fournissent pas d'informations fiables sur les sessions réellement connectées à la base de données.

Utiliser le terminal interactif PostgreSQL (psql) pour accéder à Aurora DSQL

AWS CloudShell À utiliser pour accéder à Aurora DSQL avec le terminal interactif PostgreSQL (psql)

Utilisez la procédure suivante pour accéder à Aurora DSQL avec le terminal interactif PostgreSQL depuis. AWS CloudShell Pour plus d'informations, voir Qu'est-ce que AWS CloudShell.

Pour vous connecter en utilisant AWS CloudShell

1. Connectez-vous à la console Aurora DSQL.
2. Choisissez le cluster pour lequel vous souhaitez ouvrir CloudShell. Si vous n'avez pas encore créé de cluster, suivez les étapes décrites dans Étape 1 : création d'un cluster à une seule région Aurora DSQL ou Création d'un cluster multi-régions.
3. Choisissez Connect with Query Editor, puis Connect with CloudShell.
4. Choisissez si vous souhaitez vous connecter en tant qu'administrateur ou avec un rôle de base de données personnalisé.
5. Choisissez Launch in, CloudShell puis sélectionnez Exécuter dans la CloudShell boîte de dialogue suivante.

Utilisez la CLI locale pour accéder à Aurora DSQL avec le terminal interactif PostgreSQL (psql)

Utilisez `psql` une interface de l'utilitaire PostgreSQL basée sur un terminal pour saisir des requêtes de manière interactive, les envoyer à PostgreSQL et afficher les résultats des requêtes.

> **Note**
>
> Pour améliorer les temps de réponse aux requêtes, utilisez le client PostgreSQL version 17. Si vous utilisez la CLI dans un autre environnement, assurez-vous de configurer manuellement la version 3.8+ de Python et la version 14+ de psql.

Téléchargez le programme d'installation de votre système d'exploitation depuis la page des téléchargements de PostgreSQL. Pour plus d'informations*`psql`*, consultez la section Applications clientes PostgreSQL sur le site Web de PostgreSQL.

Si vous l'avez déjà AWS CLI installé, utilisez l'exemple suivant pour vous connecter à votre cluster.

```
# Aurora DSQL requires a valid IAM token as the password when connecting.
# Aurora DSQL provides tools for this and here we're using Python.
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token \
  --region us-east-1 \
  --expires-in 3600 \
  --hostname your_cluster_endpoint)

# Aurora DSQL requires SSL and will reject your connection without it.
export PGSSLMODE=require

# Connect with psql, which automatically uses the values set in PGPASSWORD and
 PGSSLMODE.
# Quiet mode suppresses unnecessary warnings and chatty responses but still outputs
 errors.
psql --quiet \
  --username admin \
  --dbname postgres \
  --host your_cluster_endpoint
```

Utilisez le pilote Aurora DSQL pour SQLTools

Le pilote Aurora DSQL pour SQLTools est une extension Visual Studio Code pour Amazon Aurora DSQL qui s'intègre à. SQLTools Il permet aux développeurs de se connecter aux bases de données Aurora DSQL et de les interroger directement depuis VS Code. Le pilote peut être installé depuis Visual Studio Marketplace et Open VSX Registry. Kiro, Cursor et autres VSCode IDEs peuvent utiliser le registre Open VSX pour installer le pilote en suivant la procédure d'installation standard décrite dans cette page.

Caractéristiques

- Authentification IAM automatique
- Opérations de base de données standard, telles que les schémas de navigation, les tables et l'exécution de requêtes SQL.

Installation

1. Ouvrez la vue Extensions.
2. Recherchez « Aurora DSQL Driver for SQLTools ».

3. Cliquez sur « Installer ».

Remarque :

L'SQLToolsextension sera automatiquement installée si elle n'est pas déjà présente.

Authentification

Dans Aurora DSQL, toutes les connexions utilisent l'authentification basée sur IAM avec des jetons limités dans le temps. Le pilote gère automatiquement l'authentification Aurora DSQL à l'aide du connecteur Aurora DSQL pour node-postgres.

Pour plus d'informations sur l'authentification dans Aurora DSQL, consultez le guide de l'utilisateur.

Création d'une connexion SQL Aurora

Conditions préalables

- Informations d'identification AWS configurées (via l'interface de ligne de commande AWS, les variables d'environnement ou les rôles IAM)

Étapes

1. Cliquez sur l' SQLTools icône dans la barre latérale gauche.
2. Dans le SQLTools volet, survolez CONNECTIONS et cliquez sur l'icône Ajouter une nouvelle connexion.
3. Dans l'onglet SQLTools Paramètres, sélectionnez le pilote Aurora DSQL dans la liste.
4. Renseignez les paramètres de connexion.
   - Région AWS
     - Facultatif : la région sera analysée à partir du point de terminaison du cluster Aurora DSQL.
     - Obligatoire lorsque seul un ID de cluster est spécifié dans le champ Cluster DSQL.
   - AWS Profile (Profil AWS)
     - Utilisé pour la génération de jetons.
     - Utilise le profil par défaut s'il n'est pas spécifié.
5. Cliquez sur le bouton « Tester la connexion » pour tester la connexion.
6. Cliquez sur Enregistrer la connexion.

Résolution des problèmes

Expiration des informations d'authentification pour les clients SQL

Les sessions établies restent authentifiées pendant une heure maximum ou jusqu'à ce qu'une déconnexion explicite ou qu'un délai d'expiration côté client ait lieu. Si de nouvelles connexions doivent être établies, un nouveau jeton d'authentification doit être généré et fourni dans le champ Mot de passe de la connexion. Toute tentative d'ouverture d'une nouvelle session (par exemple, pour répertorier de nouvelles tables ou ouvrir une nouvelle console SQL) entraîne une nouvelle tentative d'authentification. Si le jeton d'authentification configuré dans les paramètres de connexion n'est plus valide, cette nouvelle session échouera et toutes les sessions ouvertes précédemment deviendront invalides. Gardez cela à l'esprit lorsque vous choisissez la durée de votre jeton d'authentification IAM avec l'`expires-in`option, qui peut être définie sur 15 minutes par défaut et sur une valeur maximale de sept jours.

Consultez également la section Dépannage de la documentation d'Aurora DSQL.

Outils de connectivité de cluster Amazon Aurora DSQL

Aurora DSQL est compatible avec de nombreux pilotes de base de données tiers et bibliothèques ORM. AWS fournit deux types d'outils pour simplifier l'utilisation d'Aurora DSQL :

- Connecteurs : plug-ins d'authentification qui étendent les pilotes de base de données pour gérer automatiquement la génération de jetons IAM. Utilisez des connecteurs lorsque vous travaillez directement avec des pilotes de base de données.
- Adaptateurs et dialectes : extensions pour des frameworks ORM spécifiques qui fournissent une authentification IAM et une compatibilité améliorée avec Aurora DSQL. Utilisez des adaptateurs lorsque vous travaillez avec un framework ORM pris en charge.

Adaptateurs et dialectes Aurora DSQL

Le tableau suivant indique les adaptateurs et les dialectes disponibles pour Aurora DSQL.

Langage de programmation	ORM/Cadre	Lien de référentiel
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	Tortue ORM	https://github.com/awslabs/aurora-dsql-orms//tree/main/python/tortoise-formulaire

Exemples de pilotes de base de données

Le tableau suivant présente un exemple de code pour la connexion à Aurora DSQL à l'aide de pilotes de base de données tiers.

Langage de programmation	Pilote	Exemple de lien de référentiel
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

Langage de programmation	Pilote	Exemple de lien de référentiel
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/tree/main/javascript/node-postgres
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	asyncpg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/asyncpg
Python	Psycopie	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg 2	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg2
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

Exemples d'ORM et de framework

Le tableau suivant présente un exemple de code pour l'utilisation de bibliothèques et de frameworks ORM tiers avec Aurora DSQL.

Langage de programmation	ORM/Cadre	Exemple de lien de référentiel
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate/examples/pet-clinic-app
Java	Liquibase	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/liquibase
Java	Spring Boot	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/spring_botte
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django/examples/pet-clinic-app
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy/examples/pet-clinic-app
Python	Tortue 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

Langage de programmation	ORM/Cadre	Exemple de lien de référentiel
TypeScript	Prisme	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/prisma
TypeScript	Séqueliser	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	Type ORM	https://github.com/aws-samples/aurora-dsql-samples//tree/main/typescript/type-formulaire

Chargement de données dans Aurora DSQL

Que vous effectuiez une migration depuis une base de données existante, que vous importiez des fichiers depuis Amazon Simple Storage Service ou que vous chargiez des données depuis votre système local, Aurora DSQL propose plusieurs approches pour intégrer vos données. Cette section décrit les outils et techniques recommandés pour les chargements de données de toutes tailles, des gigaoctets aux centaines de téraoctets.

Choisir une approche de chargement

Aurora DSQL prend en charge les commandes de chargement de données PostgreSQL standard, mais le chargement efficace des données à grande échelle nécessite de gérer la parallélisation, la gestion des connexions et la récupération des erreurs. Le tableau suivant récapitule vos options :

Approche	Idéal pour	Considérations
Aurora DSQL Loader - Utilitaire open source qui facilite la parallélisation des inserts lors	La plupart des scénarios de chargement de données, en particulier les migrations et les importations groupées	Gère automatiquement la parallélisation, le regroupement des connexions, la résolution des conflits et l'authentification IAM. Disponible sous forme de code source ou binaire.

Approche	Idéal pour	Considérations
de l'utilisation d'Aurora DSQL		
`\copy`PostgreSQL - Méta-commande côté client `psql`	Chargements simples lorsque vous êtes déjà connecté via `psql`	Lit les fichiers sur le client et diffuse les données via la connexion ; vous gérez vous-même la parallélisation
`INSERT`transactions - Standard `SQL DML`	Petits ensembles de données ou encarts pilotés par des applications	Approche la plus simple mais la plus lente pour les données en masse

Pour la plupart des tâches de chargement de données, utilisez le chargeur SQL Aurora. Il gère la complexité opérationnelle liée au chargement de données dans une base de données distribuée, y compris l'exécution parallèle sur plusieurs connexions et la rétentative automatique en cas d'échec des opérations.

Chargeur SQL Aurora

L'Aurora DSQL Loader est un utilitaire de ligne de commande open source conçu pour charger efficacement des données dans des clusters Aurora DSQL. Il gère le regroupement des connexions, parallélise le transfert de données entre plusieurs travailleurs, gère les conflits et réessaie automatiquement.

Fonctions principales

Le chargeur Aurora DSQL fournit les fonctionnalités suivantes :

Chargement parallèle

Les threads de travail configurables permettent le chargement simultané des données sur plusieurs connexions pour améliorer les performances.

Regroupement de connexions

Gère un pool de connexions à votre cluster SQL Aurora, en gérant automatiquement l'authentification IAM et le cycle de vie des connexions.

Support de plusieurs formats de fichiers

Supporte les formats CSV (valeurs séparées par des virgules), TSV (valeurs séparées par des tabulations) et Apache Parquet en colonnes. Le chargeur détecte automatiquement le format de fichier en fonction de l'extension d'URI source.

Inférence de schéma automatique

Lorsqu'il est utilisé avec l'`--if-not-exists`indicateur, le chargeur peut créer automatiquement des tables avec les types de colonnes appropriés en fonction des données.

Gestion des conflits

Lorsque votre table cible comporte des contraintes uniques, configurez la manière dont le chargeur gère les conflits à l'aide de l'`--on-conflict`option suivante : ignorer les doublons, modifier les enregistrements ou renvoyer une erreur.

Tolérance aux pannes

Les nouvelles tentatives automatiques et les fonctionnalités de reprise des tâches garantissent que les chargements interrompus peuvent continuer à partir de leur point d'arrêt plutôt que de redémarrer complètement.

Sources locales et S3

Chargez des données à partir de chemins de système de fichiers locaux ou directement à partir de compartiments Amazon S3 à l'aide de S3 URIs.

Conditions préalables

Avant d'utiliser le chargeur DSQL Aurora, assurez-vous de disposer des éléments suivants :

- Un cluster Aurora DSQL actif avec un point de terminaison valide.
- AWS informations d'identification configurées via les aws configure rôles AWS CLI (), AWS Single Sign-On (aws sso login) ou IAM.
- Autorisations IAM : `dsql:DbConnectAdmin` ou `dsql:DbConnect` sur votre cluster SQL Aurora.
- Pour les sources S3, autorisations appropriées pour lire depuis le compartiment source.

Installation

Téléchargez la dernière version depuis la page GitHub des versions. Des fichiers binaires prédéfinis sont disponibles pour les plateformes courantes. Pour obtenir des instructions sur la création à partir des sources, consultez le référentiel Aurora DSQL Loader.

Exemples d'utilisation

Les exemples suivants illustrent les cas d'utilisation courants de l'Aurora DSQL Loader.

Example Chargement d'un fichier CSV local

Cet exemple charge un fichier CSV depuis votre système de fichiers local dans une table existante :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table
```

Example Chargement des données à partir d'Amazon S3

Cet exemple charge un fichier Parquet depuis un compartiment Amazon S3 :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/data.parquet \
  --table my_table
```

Example Création automatique de tables

Cet exemple crée automatiquement une nouvelle table en fonction du schéma de données :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --if-not-exists
```

Example Validation avant le chargement

Cet exemple valide votre configuration sans charger réellement de données :

```
aurora-dsql-loader load \
```

```
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --dry-run
```

Example Reprise d'un chargement interrompu

Si une opération de chargement est interrompue, vous pouvez la reprendre en utilisant l'ID de tâche de l'exécution précédente :

```
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**
>
> À la reprise, le chargeur ignore la plupart des tâches déjà effectuées, mais peut réessayer certains enregistrements. Si votre table cible comporte des contraintes uniques, utilisez l'`--on-conflict`option permettant de gérer les doublons, par exemple pour les ignorer ou `DO NOTHING` pour les remplacer. `DO UPDATE`

Options de ligne de commande

Le chargeur Aurora DSQL prend en charge les options de ligne de commande suivantes :

`--endpoint`

(Obligatoire) Le point de terminaison du cluster Aurora DSQL. Exemple : *`cluster-id`*`.dsql.`*`region`*`.on.aws`

`--source-uri`

(Obligatoire) Le chemin d'accès au fichier de données. Il peut s'agir d'un chemin de fichier local ou d'un URI S3 (par exemple,`s3://`*`bucket-name`*`/`*`file.parquet`*).

`--table`

(Obligatoire) Nom de la table cible dans votre base de données Aurora DSQL.

`--if-not-exists`

(Facultatif) Créez automatiquement la table cible si elle n'existe pas. Le chargeur déduit le schéma à partir des données.

`--dry-run`

(Facultatif) Validez la configuration et les données sans les charger réellement dans la base de données.

`--resume-job-id`

(Facultatif) Reprenez une opération de chargement précédemment interrompue en utilisant l'ID de tâche spécifié.

`--manifest-dir`

(Facultatif) Répertoire pour stocker l'état des tâches et les manifestes, utilisé pour la reprise des tâches.

`--on-conflict`

(Facultatif) Spécifie comment gérer les conflits lors de l'insertion de lignes qui enfreignent les contraintes uniques de la table cible. Les valeurs valides sont `error` (renvoyer une erreur), `do-nothing` (ignorer les lignes dupliquées) ou `do-update` (mettre à jour les lignes existantes avec de nouvelles valeurs).

Pour obtenir la liste complète des options et des paramètres de configuration supplémentaires, exécutez :

```
aurora-dsql-loader load --help
```

Bonnes pratiques

- Utilisez la méthode dry-run pour la validation : testez toujours votre configuration de chargement `--dry-run` avant de charger les données dans les tables de production.
- Définissez des contraintes uniques pour la reprise : si vous devez reprendre des chargements interrompus, définissez des contraintes uniques sur vos tables cibles et utilisez l'`--on-conflict`option pour gérer les enregistrements déjà chargés.
- Utilisez Parquet pour les grands ensembles de données : le format en colonnes de Parquet permet généralement une meilleure compression et un chargement plus rapide pour les grands ensembles de données par rapport au format CSV ou TSV.

- Préserver les répertoires des manifestes : conservez le répertoire des manifestes pour les tâches de chargement jusqu'à ce que vous confirmiez que le chargement s'est bien terminé, en permettant la reprise si nécessaire.
- Pré-créez des tables lorsque cela est possible : définissez la table cible avec des types de données de colonne et des clés primaires explicites avant de charger les données. Les schémas précréés vous permettent de contrôler la précision des types et l'indexation, ce qui se traduit généralement par de meilleures performances de requête par rapport aux schémas déduits automatiquement.

Résolution des problèmes

Erreurs d'authentification

Vérifiez que vos AWS informations d'identification sont correctement configurées et que votre identité IAM possède les `dsql:DbConnectAdmin` autorisations `dsql:DbConnect` ou les autorisations requises sur le cluster cible.

Erreurs d'accès S3

Assurez-vous que votre identité IAM dispose des autorisations de lecture S3 appropriées pour le compartiment source et les objets.

Erreurs d'inférence de schéma

Lorsque vous l'utilisez`--if-not-exists`, assurez-vous que les types de colonnes de votre fichier de données sont cohérents. La combinaison de types dans une colonne peut entraîner l'échec de l'inférence de schéma.

Erreurs clés dupliquées sur le CV

Si vous rencontrez des erreurs clés dupliquées lors de la reprise d'un chargement, ajoutez des contraintes uniques à votre table cible afin que le chargeur puisse les utiliser `ON CONFLICT DO NOTHING` pour ignorer les enregistrements déjà chargés.

Pour plus d'informations sur le dépannage, consultez le GitHub référentiel Aurora DSQL Loader.

Voies de migration

Les sections suivantes décrivent comment migrer des données depuis des systèmes sources courants vers Aurora DSQL.

Migration depuis PostgreSQL

Pour migrer des données d'une base de données PostgreSQL existante vers Aurora DSQL :

1. Exportez vos données de PostgreSQL au format CSV ou Parquet. Vous pouvez utiliser la commande `COPY` PostgreSQL pour exporter chaque table :

```
COPY my_table TO '/path/to/my_table.csv' WITH (FORMAT csv, HEADER true);
```

2. Créez la table cible dans Aurora DSQL. Vous pouvez créer le schéma manuellement ou utiliser l'`--if-not-exists`indicateur du chargeur pour déduire le schéma à partir de vos données.
3. Chargez les données exportées à l'aide du chargeur SQL Aurora :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Tip**
>
> Pour les migrations de grande envergure, pensez à exporter au format Parquet pour une meilleure compression et un chargement plus rapide. Des outils tels que DuckDB peuvent convertir efficacement des fichiers CSV en Parquet.

Migration depuis MySQL

Pour migrer des données de MySQL vers Aurora DSQL :

1. Exportez vos données de MySQL au format CSV à l'aide `SELECT INTO OUTFILE` d'un outil comme celui-ci mysqldump avec l'`--tab`option :

```
SELECT * FROM my_table
INTO OUTFILE '/path/to/my_table.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Créez la table cible dans Aurora DSQL avec les types de données compatibles PostgreSQL appropriés.
3. Chargez les données exportées à l'aide du chargeur SQL Aurora :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Note**
>
> MySQL et PostgreSQL ont des systèmes de types de données différents. Passez en revue votre schéma et ajustez les types de données selon les besoins lors de la création de tables dans Aurora DSQL.

Chargement depuis Amazon S3

Si vos données se trouvent déjà dans Amazon S3, vous pouvez les charger directement sans les télécharger sur votre système local. L'Aurora DSQL Loader prend en charge S3 de manière URIs native :

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/path/to/data.parquet \
  --table my_table
```

Assurez-vous que votre identité IAM dispose d'une `s3:GetObject` autorisation sur les objets source.

Utilisation de PostgreSQL \ copy

Si vous êtes déjà connecté à Aurora DSQL via une `psql` session qui gère l'authentification IAM, vous pouvez utiliser la `\copy` méta-commande côté client pour charger des données depuis votre système de fichiers local. Contrairement à l'`COPY`instruction côté serveur, `\copy` lit le fichier sur l'ordinateur client et diffuse les données via la connexion existante, de sorte qu'aucun accès au fichier côté serveur n'est requis. Cette approche fonctionne bien pour les chargements simples à thread unique.

Example Charger un fichier CSV avec \ copy

```
\copy my_table FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

En cas d'utilisation `\copy` directe, vous êtes responsable de :

- Gestion de la parallélisation lors du chargement de plusieurs fichiers ou de grands ensembles de données
- Gestion de la gestion des connexions et de l'actualisation des jetons d'authentification
- Implémentation d'une logique de nouvelle tentative pour les opérations ayant échoué

Bonnes pratiques pour les transactions INSERT

Lorsque vous utilisez `INSERT` des instructions pour charger des données dans Aurora DSQL, suivez les pratiques suivantes pour améliorer le débit et la fiabilité :

- Répartissez les lignes en plusieurs lignes INSERTs : regroupez plusieurs lignes dans une seule `INSERT` instruction afin de réduire les allers-retours. Par exemple, `INSERT INTO my_table VALUES (1, 'a'), (2, 'b'), (3, 'c')` est plus efficace que trois instructions distinctes.
- Utiliser des requêtes paramétrées : utilisez des instructions préparées avec une liaison de paramètres au lieu d'une concaténation de chaînes. Cela évite les risques d'injection de code SQL et permet à la base de données de réutiliser les plans de requêtes.
- Limitez les transactions : Aurora DSQL utilise un contrôle de simultanéité optimiste, de sorte que les transactions volumineuses qui touchent de nombreuses lignes sont plus susceptibles de rencontrer des conflits. Optez pour des transactions de quelques centaines de lignes plutôt que de milliers.
- Implémenter une logique de nouvelle tentative : des erreurs transitoires telles que des conflits de contrôle de simultanéité optimiste (OCC) sont attendues dans un système distribué. Implémentez un ralentissement exponentiel avec une nouvelle tentative en cas d'échec des transactions.
- Paralléliser les connexions : ouvrez plusieurs connexions et distribuez des encarts sur celles-ci. Chaque connexion peut traiter simultanément un sous-ensemble différent de données.

Dans la plupart des cas d'utilisation, le chargeur DSQL Aurora fournit une approche plus simple et plus robuste du chargement des données.

Ressources supplémentaires

- Aurora DSQL Loader activé GitHub : code source, documentation et suivi des problèmes
- Création d'un jeton d'authentification dans Amazon Aurora DSQL— En savoir plus sur les jetons d'authentification IAM pour Aurora DSQL
- Accès à Aurora DSQL avec des clients compatibles avec PostgreSQL— Connectez-vous à Aurora DSQL à l'aide de différents clients et outils

IA générative pour Aurora DSQL

Cette section fournit des instructions détaillées sur l'utilisation des outils d'IA générative avec Aurora DSQL

Serveur MCP SQL Aurora d'AWS Labs

Un serveur AWS Labs Model Context Protocol (MCP) pour Aurora DSQL

Caractéristiques

- Convertir des questions et des commandes lisibles par l'homme en requêtes SQL structurées compatibles avec Postgres et les exécuter sur la base de données Aurora DSQL configurée.
- Lecture seule par défaut, transactions activées avec `--allow-writes`
- Réutilisation des connexions entre les demandes pour améliorer les performances
- Accès intégré à la documentation, à la recherche et aux recommandations relatives aux meilleures pratiques d'Aurora DSQL

Outils disponibles

Opérations sur les bases

- readonly_query - Exécute des requêtes SQL en lecture seule sur votre cluster DSQL
- transact - Exécute des opérations d'écriture dans une transaction (obligatoire) `--allow-writes`
- get_schema - Récupère les informations du schéma d'une table

Documentation et recommandations

- dsql_search_documentation - Recherche dans la documentation SQL d'Aurora

  - Paramètres : `search_phrase` (obligatoire), `limit` (facultatif)
- dsql_read_documentation - Lit des pages de documentation DSQL spécifiques
  - Paramètres : `url` (obligatoire), `start_index` (facultatif), `max_length` (facultatif)
- dsql_recommend - Obtenez des recommandations sur les meilleures pratiques DSQL
  - Paramètres : `url` (obligatoire)

Conditions préalables

1. Un compte AWS avec un cluster SQL Aurora
2. Ce serveur MCP ne peut être exécuté que localement sur le même hôte que votre client LLM.
3. Configuration des informations d'identification AWS avec accès aux services AWS
   - Vous avez besoin d'un compte AWS doté d'un rôle incluant les autorisations suivantes :
     - `dsql:DbConnectAdmin`- Connectez-vous aux clusters DSQL en tant qu'utilisateur administrateur
     - `dsql:DbConnect`- Connectez-vous aux clusters DSQL avec des rôles de base de données personnalisés (uniquement nécessaire si vous utilisez des utilisateurs non administrateurs)
   - Configuration des informations d'identification AWS avec `aws configure` ou des variables d'environnement

Installation

Pour la plupart des outils, la mise à jour de la configuration en suivant les instructions d'installation par défaut devrait être suffisante.

Des instructions distinctes sont décrites pour Claude Code et Codex.

Installation par défaut : mise à jour du fichier de configuration MCP approprié

Utilisation de **uv**

1. Installation `uv` depuis Astral ou le GitHubREADME
2. Installez Python en utilisant `uv python install 3.10`

Configurez le serveur MCP dans la configuration de votre client MCP (Trouver le fichier de configuration MCP)

```
{
```

```
  "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 de Windows

Pour les utilisateurs de Windows, le format de configuration du serveur MCP est légèrement différent :

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

Trouver le fichier de configuration du client MCP

Pour certains des outils de développement Agentic les plus courants, vous pouvez trouver les configurations de vos clients MCP dans les chemins de fichier suivants :

- Kiro :
  - Config utilisateur : `~/.kiro/settings/mcp.json`
  - Config de l'espace de travail : `/path/to/workspace/.kiro/settings/mcp.json`
- Claude Code : reportez-vous à la section Installation du code Claude pour obtenir de l'aide détaillée sur la configuration
  - Config utilisateur : `~/.claude.json` dans `"mcpServers"`
  - Config du projet : `/path/to/project/.mcp.json`
  - Config locale : `~/.claude.json` dans `"projects" -> "path/to/project" -> "mcpServers"`
- Curseur :
  - Répertoire global : `~/.cursor/mcp.json`
  - Projet : `/path/to/project/.cursor/mcp.json`
- Codex : `~/.codex/config.toml`
  - Chaque serveur MCP est configuré avec une `[mcp_servers.<server-name>]` table dans le fichier de configuration. Reportez-vous aux instructions d'installation du Codex personnalisé
- Déformation :
  - Édition de fichiers : `~/.warp/mcp_settings.json`
  - Éditeur d'applications : `Settings > AI > Manage MCP Servers` et collez le json
- CLI pour développeurs Amazon Q : `~/.aws/amazonq/mcp.json`
- Cline : généralement un chemin VS Code imbriqué - `~/.vscode-server/path/to/cline_mcp_settings.json`

Claude Code

Conditions préalables

Important : la gestion du serveur MCP n'est disponible que via l'expérience du terminal Claude Code CLI, et non via le mode de panneau natif de VS Code.

Installez d'abord la CLI Claude Code en suivant le processus d'installation natif recommandé par Claude.

Choisir le bon oscilloscope

Claude Code propose 3 champs d'application différents : local (par défaut), projet et utilisateur, et indique le champ d'application à choisir en fonction de la sensibilité des informations d'identification et du besoin de partage. Reportez-vous à la documentation Claude Code sur les étendues d'installation MCP pour plus de détails.

1. Les serveurs locaux représentent le niveau de configuration par défaut et sont stockés `~/.claude.json` sous le chemin de votre projet. Ils vous sont tous deux privés et ne sont accessibles que dans le répertoire actuel du projet. Il s'agit de la valeur par défaut `scope` lors de la création de serveurs MCP.
2. Les serveurs dédiés au projet permettent la collaboration en équipe tout en n'étant accessibles que dans un répertoire de projet. Les serveurs concernés par le projet ajoutent un `.mcp.json` fichier dans le répertoire racine de votre projet. Ce fichier est conçu pour être intégré dans le contrôle des versions, afin de garantir que tous les membres de l'équipe ont accès aux mêmes outils et services MCP. Lorsque vous ajoutez un serveur adapté au projet, Claude Code crée ou met à jour automatiquement ce fichier avec la structure de configuration appropriée.
3. Les serveurs adaptés à l'utilisateur sont stockés `~/.claude.json` et offrent une accessibilité entre les projets, ce qui les rend disponibles pour tous les projets sur votre machine tout en préservant la confidentialité de votre compte utilisateur.

Utilisation de la CLI Claude (recommandé)

L'utilisation d'une session `claude` CLI interactive permet d'améliorer l'expérience de dépannage. C'est donc la voie recommandée.

```
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]"
```

Résolution des problèmes : utilisation de Claude Code avec Bedrock sur un autre compte AWS

Si vous avez configuré Claude Code avec un compte ou un profil Bedrock AWS distinct du profil requis pour vous connecter à votre cluster dsql, vous devrez fournir des arguments d'environnement supplémentaires :

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Modification directe dans le fichier de configuration

Claude Code nécessite une dénomination alphanumérique, nous vous recommandons donc de nommer votre serveur :`aurora-dsql-mcp-server`.

Portée locale

Mettre à jour `~/.claude.json` dans le champ spécifique au projet `mcpServers` :

```
{
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Portée du projet

Mise `/path/to/project/root/.mcp.json` à jour sur le `mcpServers` terrain :

```
{
  "mcpServers": {}
}
```

Champ d'application utilisateur

Mettre à jour `~/.claude.json` dans le champ spécifique au projet `mcpServers` :

```
{
  "mcpServers": {}
}
```

Codex

Option 1 : CLI du Codex

Si la CLI Codex est installée, vous pouvez utiliser la commande codex mcp pour configurer vos serveurs MCP.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Option 2 : config.toml

Pour un contrôle plus précis des options du serveur MCP, vous pouvez modifier manuellement le fichier de `~/.codex/config.toml` configuration. Chaque serveur MCP est configuré avec une `[mcp_servers.<server-name>]` table dans le fichier de configuration.

```
[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"
```

Vérification de l'installation

Pour Amazon Q Developer CLI, Kiro CLICLI/TUI, or Codex CLI/TUI, Claude, exécutez `/mcp` pour voir l'état du serveur MCP.

Pour l'IDE Kiro, vous pouvez également accéder à l'`MCP SERVER`Onglet du panneau Kiro qui affiche tous les serveurs MCP configurés et leurs indicateurs d'état de connexion.

Options de configuration du serveur

`--allow-writes`

Par défaut, le serveur dsql mcp n'autorise pas les opérations d'écriture (« mode lecture seule »). Toute invocation de l'outil de transaction échouera dans ce mode. Pour utiliser l'outil de transaction, autorisez les écritures en passant un `--allow-writes` paramètre.

Nous recommandons d'utiliser l'accès avec le moindre privilège lors de la connexion à DSQL. Par exemple, les utilisateurs doivent utiliser un rôle en lecture seule lorsque cela est possible. Le mode lecture seule permet de faire de son mieux côté client pour rejeter les mutations.

`--cluster_endpoint`

Il s'agit d'un paramètre obligatoire pour spécifier le cluster auquel se connecter.
Il doit s'agir du point de terminaison complet de votre cluster, par exemple
`01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`

`--database_user`

Il s'agit d'un paramètre obligatoire pour spécifier l'utilisateur sous lequel se connecter. Par exemple : `admin` ou `my_user`. Notez que les informations d'identification AWS que vous utilisez doivent être autorisées à vous connecter en tant qu'utilisateur. Pour plus d'informations sur la configuration et l'utilisation des rôles de base de données dans DSQL, consultez la section Utilisation de rôles de base de données avec des rôles IAM.

`--profile`

Vous pouvez spécifier le profil AWS à utiliser pour vos informations d'identification. Notez que cela n'est pas pris en charge pour l'installation de docker.

L'utilisation de la variable d'`AWS_PROFILE`environnement dans votre configuration MCP est également prise en charge :

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Si aucun des deux n'est fourni, le serveur MCP utilise par défaut le profil « par défaut » dans votre fichier de configuration AWS.

`--region`

Il s'agit d'un paramètre obligatoire pour spécifier la région de votre base de données DSQL.

`--knowledge-server`

Paramètre facultatif permettant de spécifier le point de terminaison du serveur MCP distant pour les outils de connaissances DSQL (recherche de documentation, lecture et recommandations). Par défaut, il est préconfiguré.

Exemple :

```
--knowledge-server https://custom-knowledge-server.example.com
```

Remarque : Pour des raisons de sécurité, utilisez uniquement des points de terminaison de serveur de connaissances fiables. Le serveur doit être un point de terminaison HTTPS.

`--knowledge-timeout`

Paramètre facultatif pour spécifier le délai d'expiration en secondes pour les demandes adressées au serveur de connaissances.

Valeur par défaut : `30.0`

Exemple :

```
--knowledge-timeout 60.0
```

Augmentez cette valeur si vous rencontrez des délais d'attente lors de l'accès à la documentation sur des réseaux lents.

Pilotage SQL d'Aurora : compétences et pouvoirs

Cette section décrit comment configurer le pilotage par IA pour Aurora DSQL à l'aide de compétences et de pouvoirs. Ces fichiers de configuration basés sur le markdown fournissent un contexte et des conseils que les assistants d'intelligence artificielle appliquent automatiquement lors de la génération de code afin d'améliorer la qualité du développement agentic.

Présentation de

Les compétences et les pouvoirs sont des fonctionnalités modulaires qui étendent les fonctionnalités de l'assistant AI pour Aurora DSQL. Ils contiennent des instructions, des métadonnées et des ressources que les assistants d'intelligence artificielle utilisent automatiquement lorsqu'ils travaillent avec des bases de données Aurora DSQL.

Pourquoi utiliser les compétences et les pouvoirs

Les compétences et les pouvoirs offrent plusieurs avantages clés pour le développement d'Aurora DSQL :

- Assistants spécialisés en intelligence artificielle : fournissez une expertise spécifique à un domaine pour Aurora DSQL, notamment les meilleures pratiques, les modèles SQL compatibles avec Postgres et les optimisations de bases de données distribuées.
- Réduisez les répétitions : créez une fois, utilisez automatiquement. Élimine le besoin de fournir à plusieurs reprises les mêmes conseils au cours de plusieurs conversations.
- Efficacité du contexte : les compétences se chargent à la demande au lieu de consommer le contexte dès le départ. L'IA charge les informations par étapes selon les besoins.
- Apprentissage continu - À mesure que les fonctionnalités d'Aurora DSQL évoluent, les assistants IA accèdent automatiquement aux modèles mis à jour lorsque les compétences sont mises à jour.

Chemins de configuration recommandés

Choisissez le chemin de configuration qui correspond à votre environnement de développement :

- the section called “Compétences CLI”(Agnostique vis-à-vis des agents)
- the section called “Kiro Power”
- the section called “Claude Skill”
- the section called “Compétence Gémeaux”
- the section called “Compétence Codex”

La compétence DSQL peut également être utilisée avec d'autres agents de codage AI en copiant le dossier de compétences dans le `skills` répertoire `rules` ou l'outil.

Compétences CLI

La compétence DSQL peut être installée à l'aide de la CLI Skills. Cette méthode de configuration indépendante de l'agent fonctionne avec la plupart des assistants de codage basés sur l'IA et vous permet d'installer la compétence sur plusieurs agents à la fois.

Configuration

Exécutez la commande suivante pour installer la compétence Aurora DSQL :

```
npx skills add awslabs/mcp --skill dsql
```

La CLI vous guidera à travers :

- Sélection des agents : choisissez les agents sur lesquels installer (Kiro, Claude Code, Cursor, Copilot, Gemini, Codex, Roo, Cline, Windsurf, etc.) OpenCode
- Champ d'installation - Choisissez entre :
  - Projet : Installation dans le répertoire en cours (intégré à votre projet)
  - Global : Installation dans le répertoire de base (disponible dans tous les projets)
- Méthode d'installation - Choisissez entre :
  - Symlink (recommandé) : source unique de vérité, mises à jour faciles
  - Copie destinée à tous les agents : copies indépendantes pour chaque agent

Gestion des skills

Vérifiez et mettez à jour vos compétences à tout moment en utilisant :

```
npx skills check
npx skills update
```

Kiro Power

Les Kiro Powers sont des packages unifiés qui associent des outils MCP à une expertise en matière de framework et à des instructions de pilotage. Chaque alimentation inclut un document d'entrée expliquant les outils MCP et les déclencheurs d'activation disponibles, la configuration du serveur MCP et des instructions supplémentaires spécifiques au flux de travail chargées à la demande.

Les pouvoirs s'activent de manière dynamique en fonction du contexte utilisateur. Plutôt que de charger tous les outils dès le départ, les pouvoirs maintiennent une utilisation de base proche de zéro jusqu'à ce que les mots clés pertinents déclenchent l'activation.

Configuration

Pour configurer la puissance Kiro pour Aurora DSQL, procédez comme suit :

1. Installation directe depuis le registre Kiro Powers
2. Une fois redirigé vers Power in the IDE, vous pouvez soit :
   - Sélectionnez le bouton Try Power. Suggéré pour les utilisateurs qui souhaitent utiliser l'IA pour guider la configuration du serveur MCP ou bénéficier d'une expérience d'intégration interactive avec Aurora DSQL pour créer un nouveau cluster.
   - Ouvrez un nouveau chat Kiro et posez des questions concernant Aurora DSQL. Vous pouvez éventuellement mettre à jour la configuration MCP avec les détails de votre cluster existant pour tester la connexion au serveur MCP afin qu'elle puisse être utilisée prête à l'emploi avec l'alimentation. L'agent Kiro activera automatiquement le pouvoir s'il identifie le pouvoir comme précieux pour accomplir la tâche de l'utilisateur.

Claude Skill

Les compétences de Claude sont des capacités modulaires qui étendent les fonctionnalités de Claude. Chaque compétence regroupe des instructions, des métadonnées et des ressources facultatives que Claude utilise automatiquement le cas échéant. Les compétences sont basées sur le système de fichiers et se chargent à la demande afin de minimiser l'utilisation du contexte.

Configuration simple avec la CLI Skills

La compétence peut être installée sur Claude Code à l'aide duthe section called "Compétences CLI". Pour spécifier uniquement Claude Code comme agent sur lequel effectuer l'installation, utilisez :

```
npx skills add awslabs/mcp --skill dsql --agent claude-code
```

Alternative : Configuration directe à l'aide d'un clone Git

La configuration alternative utilise un clone fragmenté du répertoire dsql-skill et établit un lien symbolique entre ce clone et le dossier. `~/.claude/skills/` Cela permet d'apporter des modifications à la compétence chaque fois que la compétence doit être mise à jour.

Conditions préalables

- Git installé

Étapes de configuration

1. Création d'un répertoire de dépôts de base

```
mkdir -p .dsql_skill_repos
```

2. Clonez la compétence dans Sparse depuis le référentiel MCP

Clonez uniquement le `dsql-skill` dossier (aucun autre fichier) :

```
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. Associez la compétence au répertoire des compétences

Ajoutez le répertoire des compétences (par défaut : global/scopé pour l'utilisateur) :

```
mkdir -p ~/.claude/skills
```

> **Note**
>
> Si vous souhaitez en faire une compétence adaptée au projet, utilisez plutôt le répertoire racine de `.claude/skills/` votre projet.

Ajoutez le lien symbolique :

```
ln -s "$(pwd)/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/dsql-skill"
 ~/.claude/skills/dsql-skill
```

4. Vérifiez la configuration

```
# Should show SKILL.md and other skill files
ls -la ~/.claude/skills/dsql-skill/
```

5. Vérifier l'utilisation des compétences

Une fois la compétence configurée, vous devriez avoir une nouvelle commande de compétence :`/dsql`. Vous devrez peut-être redémarrer Claude Code après avoir ajouté la compétence pour qu'elle soit détectée. Vous pouvez utiliser cette commande à partir de la CLI ou du panneau Claude Code comme vous le souhaitez.

Mettre à jour la compétence

Pour extraire les dernières modifications du référentiel, procédez comme suit :

```
cd .dsql_skill_repos/mcp
git pull
```

Structure des annuaires

Après avoir configuré une compétence globale, vous devriez voir les répertoires suivants :

```
.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**
>
> Ajoutez-le `.dsql_skill_repos/` à votre `.gitignore` si vous ne souhaitez pas le suivre. Le checkout fragmenté ne conserve que le dossier des compétences, minimisant ainsi l'utilisation du disque.

Compétence Gémeaux

Pour ajouter la compétence Aurora DSQL directement dans Gemini, choisissez une portée : `workspace` (contenue dans le projet) ou `user` (par défaut, globale) et utilisez le programme d'installation des compétences.

Configuration

```
gemini skills install https://github.com/awslabs/mcp.git --path src/aurora-dsql-mcp-server/skills/dsql-skill --scope $SCOPE
```

Remplacez `$SCOPE` par l'un `workspace` ou l'autre`user`.

Vous pouvez ensuite utiliser la commande de `/dsql` compétence avec Gemini, et Gemini détectera automatiquement quand la compétence doit être utilisée.

Compétence Codex

Utilisez l'installateur de compétences de la CLI du Codex ou du TUI à l'aide de la `$skill-installer` compétence.

Configuration

```
$skill-installer install dsql skill: https://github.com/awslabs/mcp/tree/main/src/aurora-dsql-mcp-server/skills/dsql-skill
```

Redémarrez Codex pour récupérer la compétence. La compétence peut ensuite être activée à l'aide de`$dsql`.

Commencez à utiliser l'éditeur de requêtes SQL Aurora

Avec l'éditeur de requêtes Aurora DSQL, vous pouvez vous connecter en toute sécurité à vos clusters Aurora DSQL et exécuter des requêtes SQL directement depuis la console de AWS gestion sans installer ni configurer de clients externes. Il fournit un espace de travail intuitif avec mise en évidence syntaxique intégrée, saisie automatique et assistance intelligente au code. Vous pouvez rapidement explorer les objets du schéma, développer et exécuter des requêtes SQL et afficher les résultats, le tout au sein d'une seule interface.

Cette rubrique explique les étapes de connexion à un cluster, d'exécution de requêtes, d'affichage des résultats et d'exploration de fonctionnalités avancées telles que les plans d'exécution.

> **Note**
>
> L'éditeur de requêtes est disponible dans toutes les régions où Aurora DSQL est pris en charge. Pour plus de détails sur la disponibilité régionale, consultez la section Services AWS régionaux.

Conditions préalables

Avant de commencer, vérifiez que vous respectez les conditions requises suivantes :

- Vous avez au moins un cluster Aurora DSQL disponible. Pour plus de détails sur la création de clusters, consultezÉtape 1 : création d'un cluster à une seule région Aurora DSQL.
- Le point de terminaison de votre cluster est accessible au public. L'éditeur de requête ne prend pas en charge les clusters dont l'accès public est bloqué par des politiques basées sur les ressources ou les clusters gérés via des points de terminaison VPC. Pour plus de détails sur les restrictions d'accès, consultez Blocage de l'accès public à l'aide de politiques basées sur les ressources dans Aurora DSQL etGestion et connexion aux clusters SQL Amazon Aurora à l'aide de AWS PrivateLink.
- Votre utilisateur ou rôle IAM dispose des autorisations requises pour accéder au cluster et s'y connecter. Pour plus de détails sur les autorisations, consultezUtilisation des rôles de base de données et de l'authentification IAM.

Utilisation de l'éditeur de requêtes

Ouvrez l'éditeur de requêtes

Pour ouvrir l'éditeur de requêtes

1. Ouvrez la console Aurora DSQL.
2. Dans le panneau de navigation, choisissez Query Editor (Éditeur de requête).

Sur la page Clusters, vous pouvez également sélectionner le cluster que vous souhaitez interroger et choisir Connect with Query editor pour lancer directement l'éditeur.

> **Note**
>
> L'état du travail et de la connexion ne sont pas enregistrés. Si vous quittez la console Aurora DSQL, fermez l'onglet du navigateur ou déconnectez-vous, vos connexions, le texte de votre requête et les résultats sont perdus.

Connexion à un cluster

Pour se connecter à un cluster

1. S'il n'existe aucune connexion au cluster, l'éditeur affiche Aucun cluster n'a été connecté. Choisissez Connect ou sélectionnez + (Ajouter) dans le volet Cluster Explorer pour vous connecter à un cluster existant.
2. (Facultatif) Connectez-vous à plusieurs clusters ou au même cluster en utilisant différents rôles.

Explorez les objets du cluster

L'explorateur de clusters affiche toutes les connexions de cluster disponibles et vous permet de parcourir des objets tels que des bases de données, des schémas, des tables et des vues. Il fournit également des actions courantes telles que Actualiser, Créer une table et d'autres options spécifiques au contexte.

Exécution de requêtes

Pour exécuter une requête

1. Dans le volet de l'onglet de l'éditeur de requêtes, entrez votre instruction SQL. Par exemple :

   ```
   SELECT * FROM public.orders LIMIT 10;
   ```

2. Vérifiez le contexte du cluster actif affiché en haut à droite de l'onglet de requête. Cela indique la connexion au cluster associée à l'onglet de requête actuel.
3. (Facultatif) Utilisez le menu déroulant des connexions pour passer en revue toutes les connexions disponibles ou passer à un autre cluster. La modification de la connexion met à jour l'endroit où vos requêtes sont exécutées dans cet onglet.
4. Choisissez Exécuter pour exécuter la requête.

> **Note**
>
> Chaque requête peut renvoyer jusqu'à 10 000 lignes dans le volet des résultats. Pour les ensembles de données plus volumineux, affinez votre requête à l'aide de filtres ou de limites.

Examiner les résultats et les plans d'exécution

Une fois la requête exécutée, passez en revue le résultat dans le panneau Résultats en bas de l'éditeur. Par défaut, chaque exécution de requête affiche l'onglet Résultats (table), qui affiche le résultat de la requête sous forme de tableau.

Pour obtenir le plan d'exécution des requêtes, exécutez `EXPLAIN ANALYZE` ou obtenez `EXPLAIN ANALYZE VERBOSE` des informations supplémentaires sur les performances des requêtes. Pour plus de détails sur les plans d'exécution, consultezLecture des plans Aurora SQL EXPLAIN.

> **Tip**
>
> La `EXPLAIN ANALYZE VERBOSE` commande affiche les estimations d'utilisation du DPU, y compris les valeurs de calcul, de lecture, d'écriture et de total du DPU, offrant ainsi une visibilité immédiate sur les ressources consommées par les instructions SQL individuelles.

Éditeurs de requêtes : utilisation JupyterLab avec Aurora DSQL

Ce guide fournit des step-by-step instructions sur la façon de connecter et d'interroger Amazon Aurora DSQL à l' JupyterLab aide de Python. JupyterLab est un environnement informatique interactif populaire qui combine du code, du texte et des visualisations dans un seul document. Il est largement utilisé pour les applications de science des données et de recherche.

Les instructions ci-dessous décrivent les principes de base de l'utilisation d'Aurora DSQL dans le cadre d'une installation locale et dans le JupyterLab cadre de l'utilisation d'Amazon SageMaker AI, un service d'apprentissage automatique entièrement géré qui fournit un environnement hébergé avec une interface utilisateur pour les flux de travail de données.

Prise en main

Exigences

- Un cluster SQL Aurora
- Informations d'identification AWS configurées (installation locale uniquement)
- Python version 3.9 ou supérieure (installation locale uniquement)

Utilisation du local JupyterLab

Pour commencer JupyterLab, les utilisateurs doivent d'abord installer l'application en utilisant le pip de Python :

```
pip install jupyterlab
```

JupyterLab peut ensuite être ouvert en courant**`jupyter lab`**. Cela ouvrira l' JupyterLabapplication à l'adresse localhost:8888, accessible dans un navigateur. Assurez-vous que les informations d'identification AWS sont configurées dans votre environnement local avant de continuer.

Utilisation d'Amazon SageMaker AI

Dans la console AWS, accédez à la page de la console Amazon SageMaker AI, puis à la section Carnets de notes sous Applications et IDEs. À partir de là, vous pouvez sélectionner Créer une instance de bloc-notes pour commencer à créer un SageMaker environnement. Sélectionnez un type d'instance et une plate-forme avant de cliquer sur Créer une instance de bloc-notes.

Consultez la documentation de configuration d'Amazon SageMaker AI pour plus d'informations sur les options de configuration et d'instance.

> **Note**
>
> Avertissement : l'utilisation d'Amazon SageMaker AI peut entraîner des frais sur votre compte AWS.

Une fois que l' SageMaker instance est active, vous pouvez l'ouvrir depuis la section Instances du bloc-notes avec Ouvrir JupyterLab. Avant de commencer à utiliser Aurora DSQL dans votre bloc-notes, vous devez fournir un accès à votre cluster DSQL dans le rôle IAM de l' SageMaker instance.

La méthode la plus simple consiste à suivre le lien vers le rôle IAM sur la page de l'instance du bloc-notes. À partir de là, vous pouvez modifier les politiques associées à votre rôle SageMaker IAM. Consultez Authentification et autorisation pour plus d'informations sur la configuration d'une politique IAM pour autoriser l'accès à Aurora DSQL.

Connexion à Aurora DSQL à l'aide de JupyterLab

Une fois que vous avez configuré une JupyterLab instance, les étapes pour vous connecter à Aurora DSQL sont les mêmes en local et dans SageMaker AI. Créez un bloc-notes Python 3 vide, dans lequel vous pouvez ajouter des cellules avec du code Python.

Dans une cellule Python, téléchargez le certificat racine Amazon depuis le trust store officiel :

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Pour vous connecter à Aurora DSQL, installez d'abord le connecteur Aurora DSQL pour Python et le pilote Psycopg dans une cellule Python, puis importez-le :

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Une fois le connecteur importé, vous pouvez créer une configuration DSQL et vous connecter. Le connecteur Python Aurora DSQL gère automatiquement la création d'un jeton d'authentification à chaque connexion.

```
config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin"
    }

conn = dsql.connect(**config)
```

Après avoir exécuté votre code, vous devriez maintenant disposer d'une connexion Psycopg à Aurora DSQL. Vous pouvez ensuite exécuter des requêtes à l'aide du curseur Psycopg et en fournissant

votre requête SQL. Consultez la documentation de Psycopg pour plus d'informations sur l'utilisation de Psycopg avec une base de données compatible avec Postgres. Cette requête donnera lieu à une liste de tuples. `results_list`

```
with conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

Vous pouvez ensuite utiliser des frameworks Python tels que Pandas pour analyser ou visualiser les résultats de vos requêtes, par exemple :

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

Exemple de carnet

Un bloc-notes d'exemple utilisant Aurora DSQL est disponible dans le référentiel d'exemples Aurora DSQL.

Suggestions de lecture

Documentation de configuration d'Amazon SageMaker AI

Connecteur SQL Aurora pour Python

Documentation sur les pandas

Sauvegarde et restauration pour Amazon Aurora DSQL

Amazon Aurora DSQL vous aide à répondre aux exigences de conformité réglementaire et de continuité d'activités grâce à l'intégration dans AWS Backup, un service de protection des données entièrement géré qui vous permet de centraliser et d'automatiser facilement les sauvegardes sur les services AWS, dans le cloud et sur site. Le service rationalise la création, la gestion et la restauration des sauvegardes pour les clusters Aurora DSQL à une seule région ou multi-régions.

Les principales fonctionnalités sont décrites ci-après.

- Gestion centralisée des sauvegardes via la AWS Management Console le kit SDK ou l'AWS CLI
- Sauvegardes complètes de cluster
- Planifications de sauvegarde automatiques et politiques de rétention
- Capacités inter-régions et inter-comptes
- Configuration WORM (écriture unique, lecture multiple) pour toutes les sauvegardes que vous stockez

Pour plus d'informations sur les fonctionnalités d'AWS Backup Vault Lock et une liste complète des fonctionnalités d'AWS Backup disponibles pour Aurora DSQL, consultez la section Avantages de Vault Lock et disponibilité des fonctionnalités AWS Backup dans le guide du développeur AWS Backup.

Démarrer avec AWS Backup

AWS Backup crée des copies complètes de vos clusters Aurora DSQL. Vous pouvez commencer à utiliser AWS Backup pour Aurora DSQL en suivant les étapes décrites dans Mise en route avec AWS Backup :

1. Créez des sauvegardes à la demande pour une protection immédiate.
2. Établissez des plans de sauvegarde pour les sauvegardes automatisées et planifiées.
3. Configurez les périodes de conservation et la copie entre plusieurs régions.
4. Configurez la surveillance et les notifications pour les activités de sauvegarde.

Restauration de vos sauvegardes

Lorsque vous restaurez des clusters Aurora DSQL, AWS Backup crée toujours de nouveaux clusters pour préserver vos données sources.

Restauration de clusters à une seule région

Pour restaurer un cluster Aurora DSQL à une seule région, utilisez la console : https://console.aws.amazon.com/backup ou l'interface de ligne de commande (CLI) pour sélectionner le point de restauration (sauvegarde) que vous souhaitez restaurer. Configurez les paramètres du nouveau cluster qui seront créés à partir de votre sauvegarde. Pour obtenir des instructions détaillées, consultez Restauration d'un cluster Aurora DSQL à une seule région.

Restauration de clusters multi-régions

La restauration d'un cluster Aurora DSQL multi-régions est prise en charge à la fois via la console : https://console.aws.amazon.com/backup et l'AWS CLI. Pour obtenir des instructions détaillées, consultez Restauration d'un cluster Aurora DSQL multi-régions.

Pour effectuer une restauration dans un cluster Aurora DSQL multi-régions, vous pouvez utiliser une sauvegarde effectuée dans une seule Région AWS. Toutefois, avant de lancer le processus de restauration, vous devez vous assurer qu'il existe une copie identique de votre sauvegarde dans toutes les Régions AWS pour tous vos clusters multi-régions. Si vous ne possédez pas encore ces copies, vous devez d'abord copier la sauvegarde vers une autre Région AWS qui prend en charge les clusters multi-régions.

Nous vous recommandons de créer des copies de sauvegarde dans des Régions AWS clés afin de mettre en place des options de reprise après sinistre robustes et de répondre aux exigences de conformité. Pour voir les Régions AWS disponibles pour Aurora DSQL, consultez the section called "Région AWS disponibilité".

Pour obtenir des instructions détaillées sur ces étapes, consultez la documentation relative à la restauration Amazon Aurora DSQL.

Surveillance et conformité

AWS Backup fournit une visibilité complète des opérations de sauvegarde et de restauration grâce aux ressources suivantes.

- Un tableau de bord centralisé pour le suivi des tâches de sauvegarde et de restauration
- Intégration avec CloudWatch et CloudTrail.
- AWS Backup Audit Manager pour les rapports de conformité et les audits.

Consultez Journalisation des opérations Aurora DSQL à l'aide d'AWS CloudTrail pour en savoir plus sur la journalisation des enregistrements des actions réalisées par un utilisateur, un rôle ou Service AWS lors de l'utilisation d'Aurora DSQL.

Ressources supplémentaires

Pour en savoir plus sur les fonctionnalités AWS Backup et pour l'utiliser en tandem avec Aurora DSQL, consultez les ressources suivantes :

- Politiques gérées pour AWS Backup
- Restauration Amazon Aurora DSQL
- Services pris en charge par Région AWS
- Chiffrement pour les sauvegardes dans AWS Backup

En utilisant AWS Backup pour Aurora DSQL, vous mettez en œuvre une stratégie de sauvegarde robuste, conforme et automatisée qui protège les ressources critiques de votre base de données tout en minimisant les frais administratifs. Que vous gériez un seul cluster ou un déploiement multi-régions complexe, AWS Backup fournit les outils dont vous avez besoin pour garantir la sécurité et la restauration de vos données.

Surveillance et journalisation pour Aurora DSQL

La surveillance et la journalisation sont des enjeux importants pour assurer la fiabilité, la disponibilité et les performances de vos ressources Amazon Aurora DSQL. Vous devez surveiller et recueillir les données de journalisation de toutes les parties de vos ressources Aurora DSQL de telle sorte que vous puissiez déboguer facilement une défaillance à plusieurs points.

- Amazon CloudWatch surveille vos AWS ressources et les applications que vous utilisez AWS en temps réel. Vous pouvez collecter et suivre les métriques, créer des tableaux de bord personnalisés, et définir des alarmes qui vous informent ou prennent des mesures lorsqu'une métrique spécifique atteint un seuil que vous spécifiez. Par exemple, vous pouvez CloudWatch suivre l'utilisation du processeur ou d'autres indicateurs de vos EC2 instances Amazon et lancer automatiquement de nouvelles instances en cas de besoin. Pour plus d'informations, consultez le guide de CloudWatch l'utilisateur Amazon.
- AWS CloudTrailcapture les appels d'API et les événements associés effectués par vous ou en votre nom Compte AWS et envoie les fichiers journaux dans un compartiment Amazon S3 que vous spécifiez. Vous pouvez identifier les utilisateurs et les comptes appelés AWS, l'adresse IP source à partir de laquelle les appels ont été effectués et la date des appels. Pour de plus amples informations, veuillez consulter le Guide de l'utilisateur AWS CloudTrail.

Surveillance d'Aurora DSQL avec Amazon CloudWatch

Surveiller Aurora DSQL à l'aide de CloudWatch, qui recueille des données brutes et les transforme en métriques lisibles et disponibles pratiquement en temps réel. CloudWatch conserve ces statistiques pendant 15 mois, ce qui vous permet de mieux comprendre les performances de votre application Web ou de votre service. Définissez des alarmes pour surveiller des seuils spécifiques et envoyez des notifications ou prenez des mesures lorsque ces seuils sont atteints. Consultez les métriques Utilisation et observabilité suivantes disponibles pour Aurora DSQL.

Pour plus d'informations, consultez le Guide de l'utilisateur Amazon CloudWatch.

Observabilité et performance

Ce tableau présente les métriques d'observabilité pour Aurora DSQL. Il inclut des métriques pour le suivi des transactions en lecture seule et du total des transaction afin de fournir une caractérisation globale de la charge de travail. Des métriques exploitables telles que les délais d'expiration des

requêtes et le taux de conflit OCC sont inclus pour aider à identifier les problèmes de performances et les conflits de simultanéité. Les métriques liées aux sessions, à la fois actives et totales, donnent un aperçu de la charge actuelle du système.

Nom de la métrique CloudWatch	Métrique	Unité	Description
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

Métriques d'utilisation

Aurora DSQL mesure toutes les activités basées sur les demandes, telles que le traitement des requêtes, les lectures et les écritures, à l'aide d'une seule unité de facturation normalisée appelée Unité de traitement distribué (DPU).

Nom de la métrique CloudWatch	Métrique	Dimension : ResourceId	Unité	Description
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

Nom de la métrique CloudWatch	Métrique	Dimension : ResourceId	Unité	Description
				component of your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

Journalisation des opérations Aurora DSQL à l'aide d'AWS CloudTrail

Amazon Aurora DSQL est intégré à AWS CloudTrail, un service qui fournit un registre des actions prises par un utilisateur, un rôle ou un Service AWS. Il existe deux types d'événements dans CloudTrail : les événements de gestion et les événements de données. Des événements de gestion sont émis pour auditer les modifications de configuration des ressources AWS. Les événements de données capturent l'utilisation des ressources AWS généralement dans le plan de données du service.

CloudTrail capture les appels d'API pour Aurora DSQL en tant qu'événements. Aurora DSQL enregistre l'activité de la console sous forme d'événements de gestion. Aurora DSQL capture également les tentatives de connexion authentifiées aux clusters sous forme d'événements de données.

En utilisant les informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été envoyée à Aurora DSQL, l'adresse IP depuis laquelle la demande a été faite et à quelle moment elle a été faite, l'identité de l'utilisateur qui a fait la demande et d'autres détails.

CloudTrail est actif par défaut dans votre Compte AWS lorsque vous créez le compte et vous avez accès à l'historique des événements CloudTrail. L'historique des événements de CloudTrail permet

de visualiser, de rechercher, de télécharger et d'enregistrer de façon immuable les événements de gestion enregistrés au cours des 90 derniers jours dans un Région AWS. Pour plus d'informations, consultez Travailler avec l'historique des événements CloudTrail dans le AWS CloudTrailGuide de l'utilisateur. L'enregistrement de l'historique des événements ne génère aucuns frais CloudTrail.

Pour créer un enregistrement continu des événements de votre compte AWS, y compris des événements pour Aurora DSQL, créez un journal ou un magasin de données d'événements AWS CloudTrail Lake (une solution centralisée de stockage et d'analyse des événements AWS CloudTrail). Pour plus d'informations sur la création de journaux de suivi, consultez Utilisation des journaux de suivi CloudTrail. Pour plus d'informations sur la configuration et la gestion des magasins de données d'événements, consultez Magasins de données d'événements CloudTrail Lake.

Événements de gestion Aurora DSQL dans CloudTrail

Les événements de gestion CloudTrail fournissent des informations sur les opérations de gestion exécutées sur les ressources de votre compte AWS. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail capture les événements de gestion dans l'historique des événements.

Amazon Aurora DSQL journalise toutes les opérations du plan de contrôle Aurora DSQL en tant qu'événements de gestion. Pour obtenir la liste des opérations de plan de contrôle Amazon Aurora DSQL qu'Aurora DSQL journalise dans CloudTrail, consultez la référence de l'API Aurora DSQL.

Journaux de plan de contrôle

Amazon Aurora DSQL journalise les opérations du plan de contrôle Aurora DSQL suivantes vers CloudTrail en tant qu'événements de gestion.

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

- UpdateCluster

Journaux de sauvegarde et de restauration

Amazon Aurora DSQL journalise les opérations de sauvegarder et de restauration Aurora DSQL suivantes vers CloudTrail en tant qu'événements de gestion.

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

Pour en savoir plus sur la protection de vos clusters Aurora DSQL à l'aide d'AWS Backup, consultez Sauvegarde et restauration pour Amazon Aurora DSQL.

Journaux AWS KMS

Amazon Aurora DSQL journalise les opérations AWS KMS Aurora DSQL suivantes vers CloudTrail en tant qu'événements de gestion.

- `GenerateDataKey`
- `Decrypt`

Pour en savoir plus sur la façon dont les journaux CloudTrail suivent les demandes qu'Aurora DSQL envoie à AWS KMS en votre nom, consultez Surveillance de l'interaction SQL d'Aurora avec AWS KMS.

Événements de données Aurora DSQL dans CloudTrail

Les événements de données CloudTrail fournissent généralement des informations sur les opérations de ressource exécutées sur ou dans une ressource. Ils sont également utilisés pour capturer les opérations du plan de données du service. Les événements de données sont souvent des activités dont le volume est élevé. Par défaut, CloudTrail ne journalise pas les événements de données. L'historique des événements CloudTrail n'enregistre pas les événements de données.

Pour plus d'informations sur la façon de journaliser les événements de données, consultez Journalisation des événements de données avec la AWS Management Console et Journalisation des événements de données avec l'AWS Command Line Interface dans le Guide de l'utilisateur AWS CloudTrail.

Des frais supplémentaires s'appliquent pour les événements de données. Pour en savoir plus sur la tarification CloudTrail, consultez Tarification d'AWS CloudTrail.

Pour Aurora DSQL, CloudTrail capture toute tentative de connexion effectuée avec un cluster Aurora DSQL en tant qu'événement de données. Le tableau suivant répertorie les types de ressources Aurora DSQL pour lesquels vous pouvez journaliser les événements de données. La colonne Type de ressource (console) indique la valeur à choisir dans la liste Type de ressource de la console CloudTrail. La colonne resources.type value indique la valeur de `resources.type`, que vous devez spécifier lors de la configuration des sélecteurs d'événements avancés à l'aide de l'AWS CLI ou des API CloudTrail. La colonne API de données journalisées dans CloudTrail indique les appels d'API journalisés dans CloudTrail pour le type de ressource.

Type de ressource (console)	valeur resources.type	API de données journalisées dans CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Vous pouvez configurer des sélecteurs d'événements avancés pour filtrer les champs `eventName` et `resources.ARN` afin de ne journaliser que les événements filtrés. Pour plus d'informations sur ces champs, consultez AdvancedFieldSelector dans la Référence d'API AWS CloudTrail.

L'exemple suivant montre comment utiliser AWS CLI pour configurer `dsql-data-events-trail` afin de recevoir des événements de données pour Aurora DSQL.

```
aws cloudtrail put-event-selectors \
--region us-east-1 \
--trail-name dsql-data-events-trail \
--advanced-event-selectors '[{
"Name": "Log DSQL Data Events",
    "FieldSelectors": [
       { "Field": "eventCategory", "Equals": ["Data"] },
       { "Field": "resources.type", "Equals": ["AWS::DSQL::Cluster"] } ]}]'
```

Sécurité dans Amazon Aurora DSQL

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez de centres de données et d'architectures réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit ceci comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de AWS conformité Programmes de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à Amazon Aurora DSQL, consultez la section AWS Services concernés par programme de conformitéAWS .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris de la sensibilité de vos données, des exigences de votre entreprise, ainsi que de la législation et de la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation d'Aurora DSQL. Les rubriques suivantes expliquent comment configurer Aurora DSQL pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos ressources Aurora DSQL.

Rubriques

- AWS politiques gérées pour Amazon Aurora DSQL
- Protection des données dans Amazon Aurora DSQL
- Chiffrement des données pour Amazon Aurora DSQL
- Gestion des identités et des accès pour Aurora DSQL
- Politiques basées sur les ressources pour Aurora DSQL
- Utilisation de rôles liés à un service pour Aurora DSQL
- Utilisation des clés de condition IAM avec Amazon Aurora DSQL
- Réponse aux incidents dans Amazon Aurora DSQL
- Validation de la conformité pour Amazon Aurora DSQL

- Résilience dans Amazon Aurora DSQL
- Sécurité de l'infrastructure dans Amazon Aurora DSQL
- Configuration et analyse des vulnérabilités dans Amazon Aurora DSQL
- Prévention du cas de figure de l'adjoint désorienté entre services
- Bonnes pratiques de sécurité pour Aurora DSQL

AWS politiques gérées pour Amazon Aurora DSQL

Une politique AWS gérée est une politique autonome créée et administrée par AWS. AWS les politiques gérées sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

N'oubliez pas que les politiques AWS gérées peuvent ne pas accorder d'autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont accessibles à tous les AWS clients. Nous vous recommandons de réduire encore les autorisations en définissant des politiques gérées par le client qui sont propres à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les politiques AWS gérées. Si les autorisations définies dans une politique AWS gérée sont AWS mises à jour, la mise à jour affecte toutes les identités principales (utilisateurs, groupes et rôles) auxquelles la politique est attachée. AWS est le plus susceptible de mettre à jour une politique AWS gérée lorsqu'une nouvelle politique Service AWS est lancée ou lorsque de nouvelles opérations d'API sont disponibles pour les services existants.

Pour plus d'informations, consultez Politiques gérées par AWS dans le Guide de l'utilisateur IAM.

AWS politique gérée : AmazonAurora DSQLFull Accès

Vous pouvez associer `AmazonAuroraDSQLFullAccess` à vos utilisateurs, groupes et rôles.

Cette politique accorde des autorisations offrant un accès administrateur complet à Aurora DSQL. Les principaux disposant de ces autorisations peuvent :

- Créer, supprimer et mettre à jour des clusters Aurora DSQL, y compris des clusters multi-régions

- Gérer les politiques intégrées du cluster (créer, afficher, mettre à jour et supprimer des politiques)
- Ajouter et supprimer des balises des clusters
- Répertorier les clusters et afficher les informations sur les clusters individuels
- Voir les balises associées aux clusters Aurora DSQL
- Se connecter à la base de données en tant qu'utilisateur, y compris en tant qu'administrateur
- Effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, y compris le démarrage, l'arrêt et la surveillance des tâches de sauvegarde et de restauration.
- Utiliser des AWS KMS clés gérées par le client pour le chiffrement des clusters
- Afficher toutes les statistiques CloudWatch de leur compte
- Utiliser AWS Fault Injection Service (AWS FIS) pour injecter des défaillances dans les clusters SQL Aurora à des fins de test de tolérance aux pannes
- Créer des rôles liés au service pour le service `dsql.amazonaws.com`, ce qui est nécessaire pour créer des clusters

Détails de l'autorisation

Cette politique inclut les autorisations suivantes.

- `dsql` : accorde aux principaux un accès complet à Aurora DSQL.
- `cloudwatch`—accorde l'autorisation de publier des points de données métriques sur Amazon CloudWatch.
- `iam` : accorde des autorisations pour créer un rôle lié à un service.
- `backup and restore` : accorde des autorisations pour démarrer, arrêter et surveiller les tâches de sauvegarde et de restauration pour les clusters Aurora DSQL.
- `kms` : accorde les autorisations requises pour valider l'accès aux clés gérées par le client utilisées pour le chiffrement des clusters Aurora DSQL lors de la création, de la mise à jour ou de la connexion à des clusters.
- `fis`—accorde des autorisations d'utilisation AWS Fault Injection Service (AWS FIS) pour injecter des défaillances dans des clusters Aurora DSQL à des fins de tests de tolérance aux pannes.

Vous trouverez la politique `AmazonAuroraDSQLFullAccess` dans la console IAM et dans le Guide de référence de la politique gérée par AWS.

AWS politique gérée : AmazonAurora DSQLRead OnlyAccess

Vous pouvez associer `AmazonAuroraDSQLReadOnlyAccess` à vos utilisateurs, groupes et rôles.

Permet l'accès en lecture à Aurora DSQL. Les principaux disposant de ces autorisations peuvent répertorier les clusters et consulter les informations relatives à des clusters individuels. Ils peuvent voir les balises associées aux clusters Aurora DSQL et consulter les politiques intégrées des clusters. Ils peuvent récupérer et consulter toutes les statistiques CloudWatch de votre compte.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes.

- `dsql` : accorde des autorisations en lecture seule à toutes les ressources d'Aurora DSQL.
- `cloudwatch`— autorise la récupération de quantités par lots de données CloudWatch métriques et l'exécution de calculs métriques sur les données récupérées

Vous trouverez la politique `AmazonAuroraDSQLReadOnlyAccess` dans la console IAM et dans le Guide de référence de la politique gérée par AWS.

AWS politique gérée : AmazonAurora DSQLConsole FullAccess

Vous pouvez associer `AmazonAuroraDSQLConsoleFullAccess` à vos utilisateurs, groupes et rôles.

Permet un accès administratif complet à Amazon Aurora DSQL via la AWS Management Console. Les principaux disposant de ces autorisations peuvent :

- Créer, supprimer et mettre à jour des clusters Aurora DSQL, y compris des clusters multi-régions, avec la console
- Gérez les politiques intégrées du cluster via la console (création, affichage, mise à jour et suppression de politiques)
- Répertorier les clusters et afficher les informations sur les clusters individuels
- Afficher les balises de n'importe quelle ressource de votre compte

- Se connecter à la base de données en tant qu'utilisateur, y compris en tant qu'administrateur
- Effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, y compris le démarrage, l'arrêt et la surveillance des tâches de sauvegarde et de restauration.
- Utiliser des AWS KMS clés gérées par le client pour le chiffrement des clusters
- Lancement AWS CloudShell depuis le AWS Management Console
- Consultez tous les indicateurs CloudWatch de votre compte
- Utiliser AWS Fault Injection Service (AWS FIS) pour injecter des défaillances dans les clusters SQL Aurora à des fins de test de tolérance aux pannes
- Créer des rôles liés au service pour le service `dsql.amazonaws.com`, ce qui est nécessaire pour créer des clusters

Vous pouvez trouver la `AmazonAuroraDSQLConsoleFullAccess` politique sur la console IAM et AmazonAuroraDSQLConsoleFullAccessdans le AWS Managed Policy Reference Guide.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes.

- `dsql` : accorde des autorisations administratives complètes à toutes les ressources d'Aurora DSQL via la AWS Management Console.
- `cloudwatch`: autorise la récupération de quantités par lots de données CloudWatch métriques et l'exécution de calculs métriques sur les données extraites.
- `tag`—accorde l'autorisation de renvoyer les clés de balise et les valeurs actuellement utilisées dans le compte spécifié Région AWS pour le compte appelant.
- `backup and restore` : accorde des autorisations pour démarrer, arrêter et surveiller les tâches de sauvegarde et de restauration pour les clusters Aurora DSQL.
- `kms` : accorde les autorisations requises pour valider l'accès aux clés gérées par le client utilisées pour le chiffrement des clusters Aurora DSQL lors de la création, de la mise à jour ou de la connexion à des clusters.
- `cloudshell`—accorde les autorisations de lancement AWS CloudShell pour interagir avec Aurora DSQL.
- `ec2` : accorde l'autorisation de consulter les informations relatives aux points de terminaison Amazon VPC nécessaires aux connexions Aurora DSQL.

- `fis`: accorde des autorisations à utiliser AWS FIS pour injecter des défaillances dans des clusters Aurora DSQL à des fins de tests de tolérance aux pannes.
- `access-analyzer:ValidatePolicy`autorise le linter dans l'éditeur de politique, qui fournit des informations en temps réel sur les erreurs, les avertissements et les problèmes de sécurité liés à la politique actuelle.
- `fis`—accorde des autorisations d'utilisation AWS Fault Injection Service (AWS FIS) pour injecter des défaillances dans des clusters Aurora DSQL à des fins de tests de tolérance aux pannes.

Vous trouverez la politique `AmazonAuroraDSQLConsoleFullAccess` dans la console IAM et dans le Guide de référence de la politique gérée par AWS.

AWS politique gérée : Aurora DSQLService RolePolicy

Vous ne pouvez pas associer Aurora DSQLService RolePolicy à vos entités IAM. Cette politique est associée à un rôle lié au service qui permet à Aurora DSQL d'accéder aux ressources du compte.

Vous trouverez la `AuroraDSQLServiceRolePolicy` politique sur la console IAM et Aurora DSQLService RolePolicy dans le AWS Managed Policy Reference Guide.

Mises à jour des politiques AWS gérées par Aurora DSQL

Consultez les détails des mises à jour des politiques AWS gérées pour Aurora DSQL depuis que ce service a commencé à suivre ces modifications. Pour recevoir des alertes automatiques concernant les modifications apportées à cette page, abonnez-vous au flux RSS sur la Page historique du document Aurora DSQL.

Modifier	Description	Date
AmazonAuroraDSQLFullAccès et AmazonAuroraDSQLConsole FullAccess mise à jour	Ajout de la prise en charge de l'intégration AWS Fault Injection Service (AWS FIS) avec Aurora DSQL. Cela vous permet d'injecter des défaillances dans des clusters	19 août 2025

Modifier	Description	Date
	Aurora DSQL à une seule région ou multi-régions afin de tester la tolérance aux pannes de vos applications. Vous pouvez créer des modèles d'expérimentation dans la AWS FIS console pour définir des scénarios de défaillance et cibler des clusters Aurora DSQL spécifiques à des fins de test. Pour en savoir plus sur ces politiques, voir AmazonAuroraDSQLFullAccès et AmazonAuroraDSQLConsoleFullAccess.	

Modifier	Description	Date
AmazonAuroraDSQLFullAccès et AmazonAurora DSQLRead OnlyAccess AmazonAurora DSQLConsole FullAccess mise à jour	Ajout de la prise en charge des politiques basées sur les ressources (RBP) avec de nouvelles autorisations :`PutClusterPolicy` , et`GetClusterPolicy` . `DeleteClusterPolicy` Ces autorisations permettent de gérer les politiques en ligne associées aux clusters Aurora DSQL pour un contrôle d'accès précis. Pour plus d'informations, consultez AmazonAuroraDSQLFullAccess AmazonAuroraDSQLReadOnlyAccess, et AmazonAuroraDSQLConsoleFullAccess.	15 octobre 2025

Modifier	Description	Date
AmazonAuroraDSQLFullMise à jour des accès	Permet d'effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, y compris le démarrage, l'arrêt et la surveillance des tâches. Permet également d'utiliser des clés KMS gérées par le client pour le chiffrement des clusters. Pour plus d'informations, voir AmazonAuroraDSQLFullAccès et utilisation des rôles liés à un service dans Aurora DSQL.	21 mai 2025

Modifier	Description	Date
AmazonAuroraDSQLConsoleFullAccess update	Permet d'effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL via l'AWS Console Home. Cela inclut le démarrage, l'arrêt et le suivi des tâches. Cela prend également en charge l'utilisation de clés KMS gérées par le client pour le chiffrement des clusters et le lancement d'AWS CloudShell. Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLConsoleFullAccessde rôles liés à un service dans Aurora DSQL.	21 mai 2025

Modifier	Description	Date
AmazonAuroraDSQLFullMise à jour des accès	La politique ajoute quatre nouvelles autorisations pour créer et gérer des clusters de bases de données sur plusieurs Régions AWS : `PutMultiRegionProperties` `PutWitnessRegion` ,`AddPeerCluster` , et`RemovePeerCluster` . Ces autorisations incluent des contrôles au niveau des ressources et des clés de condition afin que vous puissiez contrôler les utilisateurs des clusters que vous pouvez modifier. La politique ajoute également l'autorisation `GetVpcEndpointServiceName` pour vous aider à vous connecter à vos clusters Aurora DSQL via AWS PrivateLink. Pour plus d'informations, voir Pour plus d'informations, voir AmazonAuroraDSQLFullAccès et utilisation des rôles liés à un service dans Aurora DSQL.	13 mai 2025

Modifier	Description	Date
AmazonAuroraDSQLReadOnlyAccess update	Inclut la possibilité de déterminer le nom de service de point de terminaison VPC correct lors de la connexion à vos clusters Aurora DSQL via AWS PrivateLink Aurora DSQL. Cela crée des points de terminaison uniques par cellule. Cette API vous permet donc d'identifier le point de terminaison approprié pour votre cluster et d'éviter les erreurs de connexion. Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLReadOnlyAccessde rôles liés à un service dans Aurora DSQL.	13 mai 2025

Modifier	Description	Date
AmazonAuroraDSQLConsoleFullAccess update	Ajout de nouvelles autorisations à Aurora DSQL pour prendre en charge la gestion de clusters multi-régions et la connexion des points de terminaison d'un VPC. Les nouvelles autorisations incluent : `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLConsoleFullAccessde rôles liés à un service dans Aurora DSQL.	13 mai 2025

Modifier	Description	Date
AuroraDsqlServiceLinkedRole Policy update	Permet de publier des métriques sur les espaces de noms `AWS/AuroraDSQL` et `AWS/Usage CloudWatch` dans la politique. Cela permet au service ou au rôle associé de transmettre des données d'utilisation et de performan ce plus complètes à votre CloudWatch environnement. Pour plus d'informations, reportez-vous à la section Utilisation AuroraDsqlServiceL inkedRolePolicyde rôles liés à un service dans Aurora DSQL.	8 mai 2025
Page créée	A commencé à suivre les politiques AWS gérées liées à Amazon Aurora DSQL	3 décembre 2024

Protection des données dans Amazon Aurora DSQL

Le modèle de responsabilité partagée s'applique à la protection des données dans AWS. Comme décrit dans ce modèle, AWS est responsable de la protection de l'infrastructure globale sur laquelle l'ensemble du AWS Cloud s'exécute. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des services que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée d' et RGPD (Règlement général sur la protection des données) sur le Blog de sécurité .

À des fins de protection des données, nous vous recommandons de protéger les informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou

Gestion des identités et des accès AWS. Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des sentiers pour capturer des activités, consultez Utilisation des sentiers dans le Guide de l'utilisateur.
- Utilisez des solutions de chiffrement, ainsi que tous les contrôles de sécurité par défaut au sein des Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec ou d'autres utilisateurs de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Chiffrement des données

Amazon Aurora DSQL offre une infrastructure de stockage hautement durable, pensée pour le stockage de données primaires et stratégiques. Les données sont stockées de façon redondante sur plusieurs appareils situés dans plusieurs installations au sein d'une même région Aurora DSQL.

Chiffrement en transit

Par défaut, le chiffrement en transit est configuré pour vous. Aurora DSQL utilise le protocole TLS pour chiffrer tout le trafic entre votre client SQL et Aurora DSQL.

Chiffrement et signature des données en transit entre les AWS CLI clients du SDK ou de l'API et les points de terminaison SQL Aurora :

- Aurora DSQL fournit des points de terminaison HTTPS pour le chiffrement des données en transit.
- Pour protéger l'intégrité des demandes d'API adressées à Aurora DSQL, les appels d'API doivent être signés par l'appelant. Les appels sont signés par un certificat X.509 ou par la clé d'accès AWS secrète du client conformément au processus de signature Signature version 4 (Sigv4). Pour plus d'informations, consultez Processus de signature Signature Version 4 dans le Références générales AWS.
- Utilisez le AWS CLI ou l'un des AWS SDKs pour envoyer des demandes à AWS. Ces outils signent automatiquement les demandes avec la clé d'accès que vous spécifiez lors de leur configuration.

Conformité FIPS

Les points de terminaison du plan de données Aurora DSQL (points de terminaison de cluster utilisés pour les connexions aux bases de données) utilisent par défaut des modules cryptographiques validés par la norme FIPS 140-2. Aucun point de terminaison FIPS distinct n'est requis pour les connexions au cluster.

Pour les opérations du plan de contrôle, Aurora DSQL fournit des points de terminaison FIPS dédiés dans les régions prises en charge. Pour plus d'informations sur les points de terminaison FIPS du plan de contrôle, consultez la section Points de terminaison et quotas Aurora DSQL dans le. Références générales AWS

Pour le chiffrement au repos, consultez Chiffrement au repos dans Aurora DSQL.

Confidentialité du trafic inter-réseaux

Les connexions sont protégées à la fois entre Aurora DSQL et les applications sur site et entre Aurora DSQL et les autres AWS ressources de ces applications. Région AWS

Vous disposez de deux options de connectivité entre votre réseau privé et AWS :

- Une connexion AWS Site-to-Site VPN. Pour plus d'informations, consultez Qu'est-ce qu' AWS Site-to-Site VPN ?
- Une Direct Connect connexion. Pour plus d'informations, voir Qu'est-ce que c'est Direct Connect ?

Vous accédez à Aurora DSQL via le réseau en utilisant les opérations d'API publiées par AWS. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.

- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

Protection des données dans les régions témoins

Lorsque vous créez un cluster multi-régions, une région témoin permet la reprise automatique en cas de défaillance en participant à la réplication synchrone des transactions chiffrées. Si un cluster associé devient indisponible, la région témoin reste disponible pour valider et traiter les écritures de base de données, sans perte de disponibilité.

Les régions témoins protègent et sécurisent vos données grâce à ces caractéristiques de conception :

- La région témoin reçoit et stocke uniquement les journaux de transactions chiffrés. Elle n'héberge, ne stocke ni ne transmet jamais vos clés de chiffrement.
- La région témoin se concentre uniquement sur les fonctions d'écriture, d'enregistrement des transactions et de quorum. Elle ne peut pas lire vos données de par sa conception.
- La région témoin fonctionne sans points de terminaison de connexion au cluster ni processeurs de requêtes. Cela empêche l'accès à la base de données utilisateur.

Pour plus d'informations sur les régions témoins, consultez Configuration de clusters multi-régions.

Configuration des SSL/TLS certificats pour les connexions Aurora DSQL

Aurora DSQL exige que toutes les connexions utilisent le chiffrement par protocole TLS (Transport Layer Security). Pour établir des connexions sécurisées, votre système client doit faire confiance à Amazon Root CA 1 (Root Certificate Authority). Ce certificat est préinstallé sur de nombreux systèmes d'exploitation. Cette section fournit des instructions pour vérifier le certificat Amazon Root CA 1 préinstallé sur différents systèmes d'exploitation et vous guide tout au long du processus d'installation manuelle du certificat s'il n'est pas déjà présent.

Nous vous recommandons d'utiliser la version 17 de PostgreSQL.

> **Important**
>
> Pour les environnements de production, nous recommandons d'utiliser le mode SSL `verify-full` pour garantir le plus haut niveau de sécurité de connexion. Ce mode vérifie

que le certificat du serveur est signé par une autorité de certification fiable et que le nom d'hôte du serveur correspond au certificat.

Vérification des certificats préinstallés

Dans la plupart des systèmes d'exploitation, Amazon Root CA 1 est déjà préinstallé. Pour le valider, suivez les étapes indiquées ci-dessous.

Linux (RedHat/CentOS/Fedora)

Exécutez la commande suivante dans votre terminal :

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

Si le certificat est installé, vous obtenez la sortie suivante :

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

macOS

1. Ouvrez Spotlight Search (Command + Espace)
2. Recherchez Keychain Access
3. Sélectionnez System Roots sous System Keychains
4. Recherchez Amazon Root CA 1 dans la liste des certificats

Windows

> **Note**
>
> En raison d'un problème connu avec le client Windows PSQL, l'utilisation de certificats racine du système (`sslrootcert=system`) peut renvoyer l'erreur suivante : `SSL error: unregistered scheme`. Vous pouvez suivre la méthode Connexion depuis Windows comme méthode alternative pour vous connecter à votre cluster à l'aide du protocole SSL.

Si Amazon Root CA 1 n'est pas installé sur votre système d'exploitation, suivez les étapes ci-dessous.

Installation de certificats

Si le certificat `Amazon Root CA 1` n'est pas préinstallé sur votre système d'exploitation, vous devrez l'installer manuellement afin d'établir des connexions sécurisées avec votre cluster Aurora DSQL.

Installation du certificat Linux

Suivez ces étapes pour installer le certificat Amazon Root CA sur les systèmes Linux.

1. Téléchargez le certificat racine :

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

2. Ajoutez le certificat au Trust Store :

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

3. Mettez à jour le CA Trust Store :

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

4. Vérifier l'installation :

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

Installation du certificat macOS

Ces étapes d'installation du certificat sont facultatives. L'Installation du certificat Linux fonctionne également pour un macOS.

1. Téléchargez le certificat racine :

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

2. Ajoutez le certificat au trousseau du système :

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

3. Vérifier l'installation :

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

Connexion avec SSL/TLS vérification

Avant de configurer SSL/TLS des certificats pour des connexions sécurisées à votre cluster Aurora DSQL, assurez-vous de remplir les conditions préalables suivantes.

- PostgreSQL version 17 installée
- AWS CLI configuré avec les informations d'identification appropriées
- Informations sur les points de terminaison du cluster Aurora DSQL

Connexion depuis Linux

1. Générez et définissez le jeton d'authentification :

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

2. Connectez-vous à l'aide de certificats système (s'ils sont préinstallés) :

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

3. Ou connectez-vous à l'aide d'un certificat téléchargé :

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

> **Note**
>
> Pour en savoir plus sur les paramètres PGSSLMODE, consultez sslmode dans la documentation Fonctions de contrôle de connexion à la base de données PostgreSQL 17.

Connexion depuis macOS

1. Générez et définissez le jeton d'authentification :

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

2. Connectez-vous à l'aide de certificats système (s'ils sont préinstallés) :

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

3. Vous pouvez également télécharger le certificat racine et l'enregistrer sous `root.pem` (si le certificat n'est pas préinstallé)

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

4. Connexion à l'aide de psql :

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

Connexion depuis Windows

Utilisation d'une invite de commande

1. Générez le jeton d'authentification :

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

2. Définissez la variable d'environnement mot de passe :

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

3. Définissez la configuration SSL :

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

4. Établissez une connexion à la base de données :

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

En utilisant PowerShell

1. Générez et définissez le jeton d'authentification :

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

2. Définissez la configuration SSL :

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

3. Établissez une connexion à la base de données :

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

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

Ressources supplémentaires

- Documentation PostgreSQL SSL
- Amazon Trust Services

Chiffrement des données pour Amazon Aurora DSQL

Amazon Aurora DSQL chiffre toutes les données au repos. Pour améliorer la sécurité, ce chiffrement utilise AWS Key Management Service (AWS KMS). Cette fonctionnalité réduit la lourdeur opérationnelle et la complexité induites par la protection des données sensibles. Le chiffrement au repos vous aide à :

- Réduire la charge opérationnelle liée à la protection des données sensibles
- Créer des applications sécurisées qui répondent aux exigences réglementaires et de conformité strictes en matière de chiffrement
- Ajouter une couche supplémentaire de protection des données en sécurisant toujours vos données dans un cluster chiffré
- Respecter les politiques organisationnelles, les réglementations sectorielles ou gouvernementales et les exigences de conformité

Avec Aurora DSQL, vous pouvez créer des applications sensibles en matière de sécurité qui sont conformes aux exigences réglementaires et de chiffrement strictes. Les sections suivantes expliquent comment configurer le chiffrement pour les bases de données Aurora DSQL nouvelles et existantes et comment gérer vos clés de chiffrement.

Rubriques

- Types de clés KMS pour Aurora DSQL
- Chiffrement au repos dans Aurora DSQL
- Utilisation AWS KMS et clés de données avec Aurora DSQL
- Autoriser l'utilisation de votre SQL AWS KMS key pour Aurora
- Contexte de chiffrement Aurora DSQL

- Surveillance de l'interaction SQL d'Aurora avec AWS KMS
- Création d'un cluster Aurora DSQL chiffré
- Suppression ou mise à jour d'une clé pour votre cluster Aurora DSQL
- Considérations relatives au chiffrement avec Aurora DSQL

Types de clés KMS pour Aurora DSQL

Aurora DSQL s'intègre AWS KMS pour gérer les clés de chiffrement de vos clusters. Pour plus d'informations sur les types de clés et leurs états, consultez Concepts AWS Key Management Service dans le Guide du développeur AWS Key Management Service . Lorsque vous créez un nouveau cluster, vous pouvez choisir parmi les types de clés KMS suivants pour chiffrer votre cluster :

Clé détenue par AWS

Type de chiffrement par défaut. Aurora DSQL détient la clé sans frais supplémentaires. Amazon Aurora DSQL déchiffre de manière transparente les données du cluster lorsque vous accédez à un cluster chiffré. Vous n'avez pas besoin de modifier votre code ni vos applications pour utiliser ou gérer des clusters chiffrés, et toutes les requêtes Aurora DSQL fonctionnent avec vos données chiffrées.

Clé gérée par le client

Vous créez, détenez et gérez la clé de votre Compte AWS. Vous avez le contrôle total de la clé KMS. AWS KMS des frais s'appliquent.

Le chiffrement au repos à l'aide du Clé détenue par AWS est disponible sans frais supplémentaires. Toutefois, des AWS KMS frais s'appliquent pour les clés gérées par le client. Pour plus d'informations, consultez la page Tarification d'AWS KMS.

Vous pouvez passer d'un type de clé à l'autre à tout moment. Pour plus d'informations sur les types de clés, consultez Clés gérées par le client et Clés détenues par AWS dans le Guide du développeur AWS Key Management Service .

> **Note**
>
> Le chiffrement au repos d'Aurora DSQL est disponible dans toutes les AWS régions où Aurora DSQL est disponible.

Chiffrement au repos dans Aurora DSQL

Amazon Aurora DSQL utilise la norme Advanced Encryption Standard à 256 bits (AES-256), pour chiffrer vos données au repos. Ce chiffrement permet de protéger vos données contre tout accès non autorisé au stockage sous-jacent. AWS KMS gère les clés de chiffrement de vos clusters. Vous pouvez utiliser les Clés détenues par AWS par défaut ou choisir d'utiliser vos propres AWS KMS Clés gérées par le client. Pour en savoir plus sur la spécification et la gestion des clés pour vos clusters Aurora DSQL, consultez Création d'un cluster Aurora DSQL chiffré et Suppression ou mise à jour d'une clé pour votre cluster Aurora DSQL.

Rubriques

- Clés détenues par AWS
- Clés gérées par le client

Clés détenues par AWS

Aurora DSQL chiffre tous les clusters par défaut avec. Clés détenues par AWS Ces clés sont gratuites et peuvent être modifiées chaque année afin de protéger les ressources de votre compte. Vous n'avez pas besoin de consulter, de gérer, d'utiliser ni d'auditer ces clés. Aucune action n'est donc requise pour la protection des données. Pour plus d'informations à ce sujet Clés détenues par AWS, consultez Clés détenues par AWSle guide du AWS Key Management Service développeur.

Clés gérées par le client

Vous créez, possédez et gérez les clés gérées par le client dans votre Compte AWS. Vous avez le contrôle total de ces clés KMS, notamment de leurs politiques, de leur matériel de chiffrement, de leurs balises et de leurs alias. Pour plus d'informations sur la gestion des autorisations, consultez Clés gérées par le client dans le Guide du développeur AWS Key Management Service .

Lorsque vous spécifiez une clé gérée par le client pour le chiffrement au niveau du cluster, Aurora DSQL chiffre le cluster et toutes ses données régionales avec cette clé. Pour éviter les pertes de données et maintenir l'accès au cluster, Aurora DSQL a besoin d'accéder à votre clé de chiffrement. Si vous désactivez votre clé gérée par le client, planifiez sa suppression ou si vous avez une politique qui restreint l'accès à votre service, l'état de chiffrement de votre cluster passe à `KMS_KEY_INACCESSIBLE`. Lorsqu'Aurora DSQL ne peut pas accéder à la clé, les utilisateurs ne peuvent pas se connecter au cluster, l'état de chiffrement du cluster passe à `KMS_KEY_INACCESSIBLE` et le service perd l'accès aux données du cluster.

Pour les clusters multirégionaux, les clients peuvent configurer la clé de AWS KMS chiffrement de chaque région séparément, et chaque cluster régional utilise sa propre clé de chiffrement au niveau du cluster. Si Aurora DSQL ne peut pas accéder à la clé de chiffrement d'un homologue dans un cluster multi-régions, le statut de cet homologue devient `KMS_KEY_INACCESSIBLE` et il indisponible pour les opérations de lecture et d'écriture. Les autres homologues poursuivent leurs activités normales.

> **Note**
>
> Si Aurora DSQL ne peut pas accéder à votre clé gérée par le client, l'état de chiffrement de votre cluster passe à `KMS_KEY_INACCESSIBLE`. Une fois que vous aurez rétabli l'accès aux clés, le service détectera automatiquement la restauration dans les 15 minutes. Pour plus d'informations, consultez Inactivité des clusters.
> Pour les clusters multi-régions, si l'accès à la clé est perdu pendant une période prolongée, le temps de restauration du cluster dépend de la quantité de données écrites alors que la clé était inaccessible.

Utilisation AWS KMS et clés de données avec Aurora DSQL

La fonctionnalité de chiffrement SQL au repos d'Aurora utilise une AWS KMS key et une hiérarchie de clés de données pour protéger les données de votre cluster.

Nous vous recommandons de planifier votre stratégie de chiffrement avant d'implémenter votre cluster dans Aurora DSQL. Si vous stockez des données sensibles ou confidentielles dans Aurora DSQL, pensez à inclure le chiffrement côté client dans votre stratégie. Vous pourrez ainsi chiffrer les données au plus près de leur origine et garantir leur protection tout au long de leur cycle de vie.

Rubriques

- Utilisation AWS KMS key de s avec Aurora SQL
- Utilisation des clés de cluster avec Aurora DSQL
- Mise en cache des clés de cluster

Utilisation AWS KMS key de s avec Aurora SQL

Le chiffrement au repos protège votre cluster Aurora DSQL sous une AWS KMS key. Par défaut, Aurora DSQL utilise une Clé détenue par AWS clé de chiffrement mutualisée créée et gérée dans un compte de service Aurora DSQL. Mais vous pouvez chiffrer vos clusters Aurora DSQL sous une clé gérée par le client dans votre Compte AWS. Vous pouvez sélectionner une autre clé KMS pour chaque cluster, même s'il participe à une configuration KMS multi-régions.

Vous sélectionnez la clé KMS pour un cluster lorsque vous créez ou mettez à jour le cluster. Vous pouvez modifier la clé KMS d'un cluster à tout moment, soit dans la console Aurora DSQL, soit en exécutant l'opération `UpdateCluster`. Le processus de commutation des clés ne provoque pas de durée d'indisponibilité ni de dégradation du service.

> **Important**
>
> Aurora DSQL ne prend en charge que les clés KMS symétriques. Vous ne pouvez pas utiliser une clé KMS asymétrique pour chiffrer vos clusters Aurora DSQL.

Une clé gérée par le client fournit les avantages suivants :

- Vous créez et gérez la clé KMS, y compris en définissant les stratégies de clé et les politiques IAM pour contrôler l'accès à la clé KMS. Vous pouvez activer et désactiver la clé KMS, activer et désactiver la rotation automatique des clés et supprimer la clé KMS lorsqu'elle n'est plus utilisée.
- Vous pouvez utiliser une clé gérée par le client avec un élément de clé importé ou dans un magasin de clés personnalisé que vous possédez et gérez.
- Vous pouvez auditer le chiffrement et le déchiffrement de votre cluster Aurora DSQL en examinant les appels d'API Aurora DSQL dans les AWS KMS journaux. AWS CloudTrail

Cependant, il Clé détenue par AWS est gratuit et son utilisation n'est pas prise en compte dans les quotas de AWS KMS ressources ou de demandes. Les clés gérées par le client sont facturées pour chaque appel d'API et des quotas AWS KMS s'appliquent à ces clés KMS.

Utilisation des clés de cluster avec Aurora DSQL

Aurora DSQL utilise le code AWS KMS key pour générer et chiffrer une clé de données unique pour le cluster, connue sous le nom de clé de cluster.

La clé de cluster est utilisée en tant que clé de chiffrement de clé. Aurora DSQL utilise cette clé de cluster pour protéger les clés de chiffrement des données utilisées pour chiffrer les données du cluster. Aurora DSQL génère une clé de chiffrement des données unique pour chaque structure sous-jacente dans un cluster, mais plusieurs éléments de cluster peuvent être protégés par la même clé de chiffrement des données.

Pour déchiffrer la clé de cluster, Aurora DSQL envoie une demande AWS KMS lorsque vous accédez pour la première fois à un cluster chiffré. Pour que le cluster reste disponible, Aurora DSQL vérifie régulièrement l'accès déchiffré à la clé KMS, même lorsque vous n'accédez pas activement au cluster.

Aurora DSQL stocke et utilise la clé de cluster et les clés de chiffrement des données en dehors de AWS KMS. Il protège toutes les clés avec les clés de chiffrement Advanced Encryption Standard (AES) et 256 bits. Ensuite, il stocke les clés chiffrées avec les données chiffrées afin qu'elles soient disponibles pour déchiffrer les données de cluster à la demande.

Si vous modifiez la clé KMS de votre cluster, Aurora DSQL rechiffre la clé de cluster existante avec la nouvelle clé KMS.

Mise en cache des clés de cluster

Pour éviter d'appeler AWS KMS chaque opération Aurora DSQL, Aurora DSQL met en cache les clés de cluster en texte clair pour chaque appelant en mémoire. Si Aurora DSQL reçoit une demande pour la clé de cluster mise en cache après 15 minutes d'inactivité, il envoie une nouvelle demande AWS KMS pour déchiffrer la clé de cluster. Cet appel capturera toutes les modifications apportées aux politiques d'accès du AWS KMS key in AWS KMS ou Gestion des identités et des accès AWS (IAM) après la dernière demande de déchiffrement de la clé de cluster.

Autoriser l'utilisation de votre SQL AWS KMS key pour Aurora

Si vous utilisez une clé gérée par le client dans votre compte pour protéger votre cluster Aurora DSQL, les politiques associées à cette clé doivent accorder à Aurora DSQL l'autorisation de l'utiliser en votre nom.

Vous avez un contrôle total des politiques sur une clé gérée par le client. Aurora DSQL n'a pas besoin d'autorisation supplémentaire pour utiliser la valeur par défaut Clé détenue par AWS afin de protéger les clusters Aurora DSQL de votre. Compte AWS

Politique de clé pour une clé gérée par le client

Lorsque vous sélectionnez une clé gérée par le client pour protéger un cluster Aurora DSQL, Aurora DSQL doit être autorisée à l'utiliser AWS KMS key au nom du principal qui effectue la sélection. Ce principal, qu'il s'agisse d'un utilisateur ou d'un rôle, doit disposer des AWS KMS key autorisations requises par Aurora DSQL. Vous pouvez fournir ces autorisations dans une stratégie de clé ou une politique IAM.

Aurora DSQL requiert au minimum les autorisations suivantes sur une clé gérée par le client :

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(pour kms: ReEncryptFrom et kms:ReEncryptTo)
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Par exemple, l'exemple de stratégie de clé suivant fournit uniquement les autorisations requises. La politique a les effets suivants :

- Autorise Aurora DSQL à utiliser le AWS KMS key dans des opérations cryptographiques, mais uniquement lorsqu'il agit pour le compte des principaux du compte autorisés à utiliser Aurora DSQL. Si les principaux spécifiés dans la déclaration de stratégie n'ont pas l'autorisation d'utiliser Aurora DSQL, l'appel échoue, même lorsqu'il provient du service Aurora DSQL.
- La clé de condition `kms:ViaService` permet l'accord d'autorisations uniquement lorsque la demande provient d'Aurora DSQL au nom des principaux répertoriés dans la déclaration de stratégie. Ces principals ne peuvent pas appeler ces opérations directement.

Avant d'utiliser un exemple de politique clé, remplacez les exemples de principes par des principes réels provenant de votre. Compte AWS

```
{
  "Sid": "Enable dsql IAM User Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": [
```

```
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:Encrypt",
    "kms:ReEncryptFrom",
    "kms:ReEncryptTo"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:dsql:ClusterId": "w4abucpbwuxx",
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
},
{
  "Sid": "Enable dsql IAM User Describe Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
}
```

Contexte de chiffrement Aurora DSQL

Un contexte de chiffrement est un ensemble de paires clé-valeur qui contiennent des données non secrètes arbitraires. Lorsque vous incluez un contexte de chiffrement dans une demande de chiffrement de données, lie AWS KMS cryptographiquement le contexte de chiffrement aux données chiffrées. Pour déchiffrer les données, vous devez transmettre le même contexte de chiffrement.

Aurora DSQL utilise le même contexte de chiffrement dans toutes les opérations AWS KMS cryptographiques. Si vous utilisez une clé gérée par le client pour protéger votre cluster Aurora DSQL, vous pouvez utiliser le contexte de chiffrement pour identifier l'utilisation de cette clé AWS KMS key dans les enregistrements et les journaux d'audit. Elle apparaît également texte brut dans les journaux tels que ceux d' AWS CloudTrail.

Le contexte de chiffrement peut également être utilisé comme condition d'autorisation dans les politiques.

Dans ses demandes adressées à AWS KMS, Aurora DSQL utilise un contexte de chiffrement avec une paire clé-valeur :

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

La paire clé-valeur identifie le cluster chiffrée par Aurora DSQL. La clé est `aws:dsql:ClusterId`. La valeur est l'identifiant du cluster.

Surveillance de l'interaction SQL d'Aurora avec AWS KMS

Si vous utilisez une clé gérée par le client pour protéger vos clusters Aurora DSQL, vous pouvez utiliser AWS CloudTrail les journaux pour suivre les demandes qu'Aurora DSQL envoie en votre AWS KMS nom.

Développez les sections suivantes pour découvrir comment Aurora DSQL utilise les AWS KMS opérations `GenerateDataKey` et`Decrypt`.

`GenerateDataKey`

Lorsque vous activez le chiffrement au repos sur un cluster, Aurora DSQL crée une clé de cluster unique. Il envoie une `GenerateDataKey` demande AWS KMS qui spécifie AWS KMS key le cluster.

L'événement qui enregistre l'opération `GenerateDataKey` est similaire à l'exemple d'événement suivant. L'utilisateur est le compte de service Aurora DSQL. Les paramètres incluent l'Amazon Resource Name (ARN) du AWS KMS key, un spécificateur de clé qui nécessite une clé de 256 bits, et le contexte de chiffrement qui identifie le cluster.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dsql.amazonaws.com"
    },
    "eventTime": "2025-05-16T18:41:24Z",
    "eventSource": "kms.amazonaws.com",
```

```
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "dsql.amazonaws.com",
    "userAgent": "dsql.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dsql:ClusterId": "w4abucpbwuxx"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
    },
    "responseElements": null,
    "requestID": "2da2dc32-d3f4-4d6c-8a41-aff27cd9a733",
    "eventID": "426df0a6-ba56-3244-9337-438411f826f4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "f88e0dd8-6057-4ce0-b77d-800448426d4e",
    "vpcEndpointId": "AWS Internal",
    "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
    "eventCategory": "Management"
}
```


Decrypt

Lorsque vous accédez à une cluster Aurora DSQL chiffré, Aurora DSQL a besoin de déchiffrer la clé de cluster pour pouvoir déchiffrer les clés situées au-dessous dans la hiérarchie. Il déchiffre ensuite les données du cluster. Pour déchiffrer la clé du cluster, Aurora DSQL envoie une `Decrypt` demande AWS KMS indiquant le nom du AWS KMS key cluster.

L'événement qui enregistre l'opération `Decrypt` est similaire à l'exemple d'événement suivant. L'utilisateur est le principal utilisateur Compte AWS qui accède au cluster. Les paramètres incluent la

clé de cluster chiffrée (sous forme de blob de texte chiffré) et le contexte de chiffrement identifiant le cluster. AWS KMS dérive l'identifiant du AWS KMS key à partir du texte chiffré.


```
{
  "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"
}
```

Création d'un cluster Aurora DSQL chiffré

Tous les clusters Aurora DSQL sont chiffrés au repos. Par défaut, les clusters utilisent une Clé détenue par AWS clé gratuite, ou vous pouvez spécifier une AWS KMS clé personnalisée. Procédez comme suit pour créer votre cluster chiffré à partir du AWS Management Console ou du AWS CLI.

Console

Pour créer un cluster chiffré dans AWS Management Console

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Clusters.
3. Choisissez Créer un cluster en haut à droite, puis sélectionnez Une seule région unique.
4. Dans les paramètres de chiffrement du cluster, choisissez l'une des options suivantes.
   - Acceptez les paramètres par défaut pour chiffrer sans Clé détenue par AWS frais supplémentaires.
   - Sélectionnez Personnaliser les paramètres de chiffrement (avancé) pour indiquer une clé KMS personnalisée. Ensuite, recherchez ou entrez l'ID ou l'alias de votre clé KMS. Vous pouvez également choisir Créer une AWS KMS clé pour créer une nouvelle clé dans la AWS KMS console.
5. Choisissez Créer un cluster.

Pour confirmer le type de chiffrement de votre cluster, accédez à la page Clusters et sélectionnez l'ID du cluster pour afficher les détails du cluster. Consultez l'onglet Paramètres du cluster. Le paramètre de clé KMS du cluster indique la clé par défaut Aurora DSQL pour les clusters qui utilisent des clés AWS détenues ou l'ID de clé pour les autres types de chiffrement.

> **Note**
>
> Si vous choisissez de posséder et de gérer votre propre clé, assurez-vous que la stratégie de clé de KMS est correctement définie. Pour plus d'informations et d'exemples, consultez the section called "Politique de clé pour une clé gérée par le client".

CLI

Pour créer un cluster chiffré avec la valeur par défaut Clé détenue par AWS

- Utilisez la commande suivante pour créer un cluster Aurora DSQL :

```
aws dsql create-cluster
```

Comme indiqué dans les détails de chiffrement suivants, l'état de chiffrement du cluster est activé par défaut, et le type de chiffrement par défaut est la clé détenue par AWS. Le cluster est désormais chiffré avec la clé détenue par AWS par défaut dans le compte de service Aurora DSQL.

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

Pour créer un cluster chiffré à l'aide de votre clé gérée par le client

- Utilisez la commande suivante pour créer un cluster Aurora DSQL, en remplaçant l'ID de clé en rouge par l'ID de votre clé gérée par le client.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Comme indiqué dans les détails de chiffrement suivants, l'état de chiffrement du cluster est activé par défaut, et le type de chiffrement est la clé KMS gérée par le client. Le cluster est désormais chiffré avec votre clé.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Suppression ou mise à jour d'une clé pour votre cluster Aurora DSQL

Vous pouvez utiliser le AWS Management Console ou AWS CLI pour mettre à jour ou supprimer les clés de chiffrement des clusters existants dans Amazon Aurora DSQL. Si vous supprimez une clé sans la remplacer, Aurora DSQL utilise la Clé détenue par AWS par défaut. Procédez comme suit pour mettre à jour les clés de chiffrement d'un cluster existant à partir de la console Aurora DSQL ou de l' AWS CLI.

Console

Pour mettre à jour ou supprimer une clé de chiffrement dans le AWS Management Console

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Dans le volet de navigation sur le côté gauche de la console, choisissez Clusters.
3. Dans la vue en liste, recherchez et sélectionnez la ligne du cluster que vous souhaitez mettre à jour.
4. Sélectionnez le menu Actions, puis Modifier.
5. Dans les paramètres de chiffrement du cluster, choisissez l'une des options suivantes pour modifier les paramètres de chiffrement.
   - Si vous souhaitez passer d'une clé personnalisée à une Clé détenue par AWS, désélectionnez l'option Personnaliser les paramètres de chiffrement (avancés). Les paramètres par défaut s'appliqueront et chiffreront votre cluster gratuitement. Clé détenue par AWS
   - Si vous souhaitez passer d'une clé KMS personnalisée à une autre ou d'une Clé détenue par AWS à une clé KMS, sélectionnez l'option Personnaliser les paramètres de chiffrement (avancé) si elle n'est pas déjà sélectionnée. Recherchez et sélectionnez ensuite l'ID ou l'alias de la clé que vous souhaitez utiliser. Vous pouvez également choisir Créer une AWS KMS clé pour créer une nouvelle clé dans la AWS KMS console.
6. Choisissez Enregistrer.

CLI

Les exemples suivants montrent comment utiliser le pour mettre AWS CLI à jour un cluster chiffré.

Pour mettre à jour un cluster chiffré avec la valeur par défaut Clé détenue par AWS

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

L'état `EncryptionStatus` de la description du cluster est défini sur `ENABLED` et le `EncryptionType` est `AWS_OWNED_KMS_KEY`.

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

Ce cluster est désormais chiffré à l'aide de la valeur par défaut Clé détenue par AWS du compte de service Aurora DSQL.

Mise à jour d'un cluster chiffré avec une clé gérée par le client pour Aurora DSQL

Mettez à jour le cluster chiffré, comme dans l'exemple suivant :

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

L'état `EncryptionStatus` de la description du cluster passe à `UPDATING` et le `EncryptionType` est `CUSTOMER_MANAGED_KMS_KEY`. Une fois qu'Aurora DSQL aura fini de propager la nouvelle clé sur la plate-forme, l'état du chiffrement passera à `ENABLED`

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Si vous choisissez de posséder et de gérer votre propre clé, assurez-vous que la stratégie de clé de KMS est correctement définie. Pour plus d'informations et d'exemples, consultez the section called "Politique de clé pour une clé gérée par le client".

Considérations relatives au chiffrement avec Aurora DSQL

- Aurora DSQL chiffre toutes les données au repos du cluster. Vous ne pouvez pas désactiver ce chiffrement ni chiffrer uniquement certains éléments d'un cluster.
- AWS Backup chiffre vos sauvegardes et tous les clusters restaurés à partir de ces sauvegardes. Vous pouvez chiffrer vos données de sauvegarde à AWS Backup l'aide de la clé AWS détenue ou d'une clé gérée par le client.
- Les états de protection des données suivants sont activés pour Aurora DSQL :
  - Données au repos : Aurora DSQL chiffre toutes les données statiques sur les supports de stockage persistants
  - Données en transit : Aurora DSQL chiffre toutes les communications à l'aide du protocole TLS (Transport Layer Security) par défaut
- Lorsque vous passez à une autre clé, nous vous recommandons de conserver la clé d'origine activée jusqu'à ce que la transition soit terminée. AWS a besoin de la clé d'origine pour déchiffrer les données avant de les chiffrer avec la nouvelle clé. Le processus est terminé lorsque l'état `encryptionStatus` du cluster est `ENABLED` et que vous voyez `kmsKeyArn` sur la nouvelle clé gérée par le client.
- Lorsque vous désactivez votre clé gérée par le client ou que vous révoquez l'accès à Aurora DSQL pour utiliser votre clé, votre cluster passe à l'état `IDLE`.
- L'API SQL AWS Management Console et Amazon Aurora utilisent des termes différents pour désigner les types de chiffrement :
  - AWS Console — Dans la console, vous verrez `KMS` quand vous utilisez une clé gérée par le client et `DEFAULT` quand vous utilisez un Clé détenue par AWS.
  - API : l'API Amazon Aurora DSQL utilise `CUSTOMER_MANAGED_KMS_KEY` pour les clés gérées par le client et `AWS_OWNED_KMS_KEY` pour les Clés détenues par AWS.
- Si vous ne spécifiez pas de clé de chiffrement lors de la création du cluster, Aurora DSQL chiffre automatiquement vos données à l'aide du. Clé détenue par AWS

- Vous pouvez à tout moment passer d'une Clé détenue par AWS clé gérée par le client à une clé gérée par le client. Effectuez cette modification à l'aide de AWS Management Console AWS CLI, ou de l'API SQL Amazon Aurora.

Gestion des identités et des accès pour Aurora DSQL

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Des administrateurs IAM contrôlent les personnes qui s'authentifient (sont connectées) et sont autorisées (disposent d'autorisations) à utiliser des ressources Aurora DSQL. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l'accès à l'aide de politiques
- Fonctionnement d'Amazon Aurora DSQL avec IAM
- Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL
- Résolution des problèmes liés à l'identité et à l'accès à Amazon Aurora DSQL

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir Résolution des problèmes liés à l'identité et à l'accès à Amazon Aurora DSQL)
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir Fonctionnement d'Amazon Aurora DSQL avec IAM)
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL)

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle utilisateur à un rôle IAM (console) ou en appelant une opération AWS CLI ou AWS API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Vous devez spécifier un principal dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.

- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Fonctionnement d'Amazon Aurora DSQL avec IAM

Avant d'utiliser IAM pour gérer l'accès à Aurora DSQL, découvrez les fonctionnalités IAM que vous pouvez utiliser avec Aurora DSQL.

Fonctionnalités IAM que vous pouvez utiliser avec Amazon Aurora DSQL

Fonctionnalité IAM	Prise en charge par Aurora DSQL
Politiques basées sur l'identité	Oui
Politiques basées sur les ressources	Oui
Actions de politique	Oui
Ressources de politique	Oui
Clés de condition de politique	Oui
ACLs	Non
ABAC (étiquettes dans les politiques)	Oui
Informations d'identification temporaires	Oui
Autorisations de principal	Oui
Rôles de service	Oui

Fonctionnalité IAM	Prise en charge par Aurora DSQL
Rôles liés à un service	Oui

Pour obtenir une vue d'ensemble de la façon dont Aurora DSQL et les autres AWS services fonctionnent avec la plupart des fonctionnalités IAM, consultez la section AWS Services compatibles avec IAM dans le Guide de l'utilisateur IAM.

Politiques basées sur l'identité pour Aurora DSQL

Prend en charge les politiques basées sur l'identité : oui

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Exemples de politiques basées sur l'identité pour Aurora DSQL

Pour voir des exemples de politiques Aurora DSQL basées sur l'identité, consultez Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL.

Politiques basées sur une ressource dans Aurora DSQL

Prend en charge les politiques basées sur les ressources : oui

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent

les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les mandataires peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou des services AWS. Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les stratégies gérées par AWS depuis IAM dans une stratégie basée sur les ressources.

Pour savoir comment créer et gérer des politiques basées sur les ressources pour les clusters Aurora DSQL, consultez la section Stratégies basées sur les ressources pour Aurora DSQL.

Actions de politique pour Aurora DSQL

Prend en charge les actions de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour afficher la liste des actions Aurora DSQL, consultez Actions définies par Amazon Aurora DSQL dans la Référence de l'autorisation de service.

Les actions de politique dans Aurora DSQL utilisent le préfixe suivant avant l'action :

```
dsql
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

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

Pour voir des exemples de politiques Aurora DSQL basées sur l'identité, consultez Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL.

Ressources de politique pour Aurora DSQL

Prend en charge les ressources de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de ressources Aurora DSQL et leurs caractéristiques ARNs, consultez la section Ressources définies par Amazon Aurora DSQL dans le Service Authorization Reference. Pour découvrir les actions avec lesquelles vous pouvez spécifier l'ARN de chaque ressource, consultez Actions définies par Amazon Aurora DSQL.

Pour voir des exemples de politiques Aurora DSQL basées sur l'identité, consultez Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL.

Clés de condition d'une politique pour Aurora DSQL

Prend en charge les clés de condition de politique spécifiques au service : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` indique à quel moment les instructions s'exécutent en fonction de critères définis. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs

de la demande. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM.

Pour afficher la liste des clés de condition Aurora DSQL, consultez Clés de condition pour Amazon Aurora DSQL dans la Référence de l'autorisation de service. Pour découvrir les actions et les ressources avec lesquelles vous pouvez utiliser une clé de condition, consultez Actions définies par Amazon Aurora DSQL.

Pour voir des exemples de politiques Aurora DSQL basées sur l'identité, consultez Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL.

ACLs dans Aurora DSQL

Supports ACLs : Non

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

ABAC avec Aurora DSQL

Prise en charge d'ABAC (balises dans les politiques) : Oui

Le contrôle d'accès par attributs (ABAC) est une stratégie d'autorisation qui définit les autorisations en fonction des attributs appelés balises. Vous pouvez associer des balises aux entités et aux AWS ressources IAM, puis concevoir des politiques ABAC pour autoriser les opérations lorsque la balise du principal correspond à la balise de la ressource.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur ABAC, consultez Définition d'autorisations avec l'autorisation ABAC dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez Utilisation du contrôle d'accès par attributs (ABAC) dans le Guide de l'utilisateur IAM.

Utilisation des informations d'identification temporaires avec Aurora DSQL

Prend en charge les informations d'identification temporaires : oui

Les informations d'identification temporaires fournissent un accès à court terme aux AWS ressources et sont automatiquement créées lorsque vous utilisez la fédération ou que vous changez de rôle. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez Informations d'identification de sécurité temporaires dans IAM et Services AWS compatibles avec IAM dans le Guide de l'utilisateur IAM.

Autorisations de principal entre services pour Aurora DSQL

Prend en charge les sessions d'accès direct (FAS) : oui

Les sessions d'accès direct (FAS) utilisent les autorisations du principal appelant et Service AWS, combinées Service AWS à la demande d'envoi de demandes aux services en aval. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez la section Sessions de transmission d'accès.

Rôles de service pour Aurora DSQL

Prend en charge les rôles de service : oui

Un rôle de service est un rôle IAM qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

> **Warning**
>
> La modification des autorisations d'un rôle de service peut altérer la fonctionnalité d'Aurora DSQL. Ne modifiez des rôles de service que quand Aurora DSQL vous le conseille.

Rôles liés à un service pour Aurora DSQL

Prend en charge les rôles liés à un service : oui

Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre

Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service pour Aurora DSQL, consultez Utilisation de rôles liés à un service pour Aurora DSQL.

Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ni à modifier les ressources Aurora DSQL. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM.

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, consultez Création de politiques IAM (console) dans le Guide de l'utilisateur IAM.

Pour plus de détails sur les actions et les types de ressources définis par Aurora DSQL, y compris le format de ARNs pour chacun des types de ressources, consultez la section Actions, ressources et clés de condition pour Amazon Aurora DSQL dans le manuel Service Authorization Reference.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la console Aurora DSQL
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations
- Autoriser la gestion du cluster et la connexion aux bases de données
- Accès aux ressources SQL Aurora basé sur des balises

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources Aurora DSQL dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à

vos cas d'utilisation. Pour plus d'informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.

- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.
- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez Validation de politiques avec IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Sécurisation de l'accès aux API avec MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de la console Aurora DSQL

Pour accéder à la console Amazon Aurora DSQL, vous devez disposer d'un ensemble minimum d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher des informations détaillées sur les ressources Aurora DSQL de votre Compte AWS. Si vous créez une politique basée

sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la console Aurora DSQL, associez également le DSQL Aurora `AmazonAuroraDSQLConsoleFullAccess` ou la politique `AmazonAuroraDSQLReadOnlyAccess` AWS gérée aux entités. Pour plus d'informations, consultez Ajout d'autorisations à un utilisateur dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
```

```
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Autoriser la gestion du cluster et la connexion aux bases de données

La politique suivante accorde à un utilisateur IAM l'autorisation de gérer et de se connecter à un cluster Aurora DSQL spécifique. La politique couvre la gestion du cluster et les actions de connexion à un seul cluster Amazon Resource Name (ARN), tout en autorisant `dsql:ListClusters` l'accès à toutes les ressources, car cette action ne prend pas en charge les autorisations au niveau des ressources.

Cet exemple permet `dsql:DbConnectAdmin` de se connecter au `admin` rôle. Pour vous connecter à un rôle de base de données personnalisé, remplacez `dsql:DbConnectAdmin` par`dsql:DbConnect`. Pour de plus amples informations, veuillez consulter Authentification et autorisation pour 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": "*"
        }
    ]
}
```

Accès aux ressources SQL Aurora basé sur des balises

Vous pouvez utiliser des conditions dans votre politique basée sur l'identité pour contrôler l'accès aux ressources SQL Aurora en fonction de balises. L'exemple suivant montre comment créer une politique permettant de visualiser un cluster. Toutefois, la politique n'accorde l'autorisation que si la balise de cluster `Owner` a la valeur du nom d'utilisateur de cet utilisateur. Cette politique accorde également les autorisations nécessaires pour réaliser cette action sur la console.

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}"
                }
            }
        }
    ]
```

```
}
```

Vous pouvez rattacher cette politique aux utilisateurs IAM de votre compte. Si un utilisateur nommé `richard-roe` tente d'afficher un cluster Aurora DSQL, le cluster doit être balisé `0wner=richard-roe` ou`owner=richard-roe`. Dans le cas contraire, IAM refuse l'accès. La clé de condition d'étiquette `0wner` correspond à la fois à `0wner` et à `owner`, car les noms de clé de condition ne sont pas sensibles à la casse. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

La politique suivante permet à un utilisateur de créer des clusters uniquement s'il étiquette le cluster avec son propre nom d'utilisateur en tant que`0wner`. Il permet également de baliser uniquement les clusters que l'utilisateur possède déjà.

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}"
                }
            }
        }
```

```
    ]
}
```

Résolution des problèmes liés à l'identité et à l'accès à Amazon Aurora DSQL

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec Aurora DSQL et IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans Aurora DSQL
- Je ne suis pas autorisé à effectuer iam : PassRole
- Je souhaite autoriser des personnes extérieures à moi Compte AWS à accéder à mes ressources Aurora DSQL

Je ne suis pas autorisé à effectuer une action dans Aurora DSQL

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit lorsque le `mateojackson` tente d'utiliser la console pour afficher les détails d'une ressource *`my-dsql-cluster`*, mais ne dispose pas des autorisations *`GetCluster`* nécessaires.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-dsql-cluster`* à l'aide de l'action *`GetCluster`*.

Si vous avez encore besoin d'aide, contactez votre administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter l'action `iam:PassRole`, vos politiques doivent être mises à jour afin de vous permettre de transmettre un rôle à Aurora DSQL.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans Aurora DSQL. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite autoriser des personnes extérieures à moi Compte AWS à accéder à mes ressources Aurora DSQL

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si Aurora DSQL prend en charge ces fonctionnalités, consultez Fonctionnement d'Amazon Aurora DSQL avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.

- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources pour Aurora DSQL

Utilisez des politiques basées sur les ressources pour Aurora DSQL afin de restreindre ou d'accorder l'accès à vos clusters par le biais de documents de politique JSON directement associés aux ressources de votre cluster. Ces politiques fournissent un contrôle précis sur les personnes autorisées à accéder à votre cluster et dans quelles conditions.

Les clusters Aurora DSQL sont accessibles depuis l'Internet public par défaut, l'authentification IAM étant le principal contrôle de sécurité. Les politiques basées sur les ressources vous permettent d'ajouter des restrictions d'accès, notamment pour bloquer l'accès depuis l'Internet public.

Les politiques basées sur les ressources fonctionnent parallèlement aux politiques basées sur l'identité IAM. AWS évalue les deux types de politiques afin de déterminer les autorisations finales pour toute demande d'accès à votre cluster. Par défaut, les clusters Aurora DSQL sont accessibles au sein d'un compte. Si un utilisateur ou un rôle IAM dispose d'autorisations Aurora DSQL, il peut accéder aux clusters sans qu'aucune politique basée sur les ressources ne soit attachée.

> **Note**
>
> Les modifications apportées aux politiques basées sur les ressources sont finalement cohérentes et prennent généralement effet en une minute.

Pour plus d'informations sur les différences entre les politiques basées sur l'identité et les politiques basées sur les ressources, voir Politiques basées sur l'identité et politiques basées sur les ressources dans le guide de l'utilisateur IAM.

Quand utiliser des politiques basées sur les ressources

Les politiques basées sur les ressources sont particulièrement utiles dans les scénarios suivants :

- Contrôle d'accès basé sur le réseau : limitez l'accès en fonction du VPC ou de l'adresse IP d'où proviennent les demandes, ou bloquez complètement l'accès public à Internet. Utilisez des clés de condition telles que `aws:SourceVpc` et `aws:SourceIp` pour contrôler l'accès au réseau.
- Plusieurs équipes ou applications : accordez l'accès au même cluster à plusieurs équipes ou applications. Plutôt que de gérer des politiques IAM individuelles pour chaque principal, vous définissez les règles d'accès une fois sur le cluster.
- Accès conditionnel complexe : contrôlez l'accès en fonction de plusieurs facteurs tels que les attributs réseau, le contexte de la demande et les attributs utilisateur. Vous pouvez combiner plusieurs conditions dans une seule politique.
- Gouvernance de sécurité centralisée : permettez aux propriétaires de clusters de contrôler l'accès à l'aide d'une syntaxe de AWS politique familière qui s'intègre à vos pratiques de sécurité existantes.

> **Note**
>
> L'accès entre comptes n'est pas encore pris en charge pour les politiques basées sur les ressources Aurora DSQL, mais il sera disponible dans les futures versions.

Lorsque quelqu'un essaie de se connecter à votre cluster Aurora DSQL, il AWS évalue votre politique basée sur les ressources dans le cadre du contexte d'autorisation, ainsi que toutes les politiques IAM pertinentes, afin de déterminer si la demande doit être autorisée ou rejetée.

Les politiques basées sur les ressources peuvent accorder l'accès aux principaux au sein du même AWS compte que le cluster. Pour les clusters multirégionaux, chaque cluster régional possède sa propre politique basée sur les ressources, permettant des contrôles d'accès spécifiques à la région en cas de besoin.

> **Note**
>
> Les clés de contexte des conditions peuvent varier d'une région à l'autre (par exemple, VPC IDs).

Rubriques

- Création de clusters avec des politiques basées sur les ressources
- Ajouter et modifier des politiques basées sur les ressources pour les clusters
- Afficher les politiques basées sur les ressources
- Supprimer les politiques basées sur les ressources
- Exemples de politiques communes basées sur les ressources
- Blocage de l'accès public à l'aide de politiques basées sur les ressources dans Aurora DSQL
- Opérations d'API SQL Aurora et politiques basées sur les ressources

Création de clusters avec des politiques basées sur les ressources

Vous pouvez associer des politiques basées sur les ressources lors de la création d'un nouveau cluster afin de garantir que les contrôles d'accès sont en place dès le départ. Chaque cluster peut avoir une seule politique intégrée directement attachée au cluster.

AWS Console de gestion

Pour ajouter une politique basée sur les ressources lors de la création du cluster

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Choisissez Créer un cluster.
3. Configurez le nom de votre cluster, les balises et les paramètres multirégionaux selon vos besoins.
4. Dans la section Paramètres du cluster, recherchez l'option de stratégie basée sur les ressources.
5. Activez l'option Ajouter une politique basée sur les ressources.
6. Entrez votre document de politique dans l'éditeur JSON. Par exemple, pour bloquer l'accès public à Internet :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
```

```
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Vous pouvez utiliser Modifier la déclaration ou Ajouter une nouvelle déclaration pour élaborer votre politique.
8. Terminez la configuration de cluster restante et choisissez Create cluster.

AWS CLI

Utilisez le `--policy` paramètre lors de la création d'un cluster pour associer une politique en ligne :

```
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());
```

Ajouter et modifier des politiques basées sur les ressources pour les clusters

AWS Console de gestion

Pour ajouter une politique basée sur les ressources à un cluster existant

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Choisissez votre cluster dans la liste des clusters pour ouvrir la page des détails du cluster.
3. Sélectionnez l'onglet Autorisations.
4. Dans la section Stratégie basée sur les ressources, choisissez Ajouter une politique.
5. Entrez votre document de politique dans l'éditeur JSON. Vous pouvez utiliser Modifier la déclaration ou Ajouter une nouvelle déclaration pour élaborer votre politique.
6. Choisissez Add policy (Ajouter la politique).

Pour modifier une politique basée sur les ressources existante

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Choisissez votre cluster dans la liste des clusters pour ouvrir la page des détails du cluster.
3. Sélectionnez l'onglet Autorisations.
4. Dans la section Stratégie basée sur les ressources, choisissez Modifier.
5. Modifiez le document de politique dans l'éditeur JSON. Vous pouvez utiliser Modifier la déclaration ou Ajouter une nouvelle déclaration pour mettre à jour votre politique.
6. Sélectionnez Enregistrer les modifications.

AWS CLI

Utilisez la `put-cluster-policy` commande pour associer une nouvelle politique ou mettre à jour une politique existante sur un cluster :

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

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);
```

Afficher les politiques basées sur les ressources

Vous pouvez consulter les politiques basées sur les ressources associées à vos clusters pour comprendre les contrôles d'accès actuellement en place.

AWS Console de gestion

Pour consulter les politiques basées sur les ressources

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Choisissez votre cluster dans la liste des clusters pour ouvrir la page des détails du cluster.
3. Sélectionnez l'onglet Autorisations.
4. Consultez la politique ci-jointe dans la section Stratégie basée sur les ressources.

AWS CLI

Utilisez la `get-cluster-policy` commande pour afficher la politique basée sur les ressources d'un cluster :

```
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());
```

Supprimer les politiques basées sur les ressources

Vous pouvez supprimer les politiques basées sur les ressources des clusters pour modifier les contrôles d'accès.

> **Important**
>
> Lorsque vous supprimez toutes les politiques basées sur les ressources d'un cluster, l'accès est entièrement contrôlé par les politiques basées sur l'identité IAM.

AWS Console de gestion

Pour supprimer une politique basée sur les ressources

1. Connectez-vous à la console de AWS gestion et ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsql/adresse.
2. Choisissez votre cluster dans la liste des clusters pour ouvrir la page des détails du cluster.
3. Sélectionnez l'onglet Autorisations.
4. Dans la section Stratégie basée sur les ressources, choisissez Supprimer.
5. Dans la boîte de dialogue de confirmation, tapez **`confirm`** pour confirmer la suppression.
6. Sélectionnez Delete (Supprimer).

AWS CLI

Utilisez la `delete-cluster-policy` commande pour supprimer une politique d'un cluster :

```
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");
```

Exemples de politiques communes basées sur les ressources

Ces exemples présentent des modèles courants de contrôle de l'accès à vos clusters Aurora DSQL. Vous pouvez combiner et modifier ces modèles pour répondre à vos besoins d'accès spécifiques.

Bloquer l'accès public à Internet

Cette politique bloque les connexions à vos clusters Aurora DSQL depuis l'Internet public (non VPC). La politique ne précise pas à partir de quel VPC les clients peuvent se connecter, mais seulement qu'ils doivent se connecter à partir d'un VPC. Pour limiter l'accès à un VPC spécifique, utilisez-le `aws:SourceVpc` avec l'opérateur de `StringEquals` condition.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> Cet exemple est utilisé uniquement `aws:SourceVpc` pour vérifier les connexions VPC. Les touches de `aws:SourceVpce` condition `aws:VpcSourceIp` et fournissent une granularité supplémentaire mais ne sont pas requises pour le contrôle d'accès de base réservé aux VPC.

Pour fournir une exception pour des rôles spécifiques, utilisez plutôt cette politique :

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

Restreindre l'accès à AWS l'organisation

Cette politique restreint l'accès aux principaux au sein d'une AWS organisation :

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

Restreindre l'accès à une unité organisationnelle spécifique

Cette politique restreint l'accès aux principaux au sein d'une unité organisationnelle (UO) spécifique d'une AWS organisation, offrant ainsi un contrôle plus précis que l'accès à l'échelle de l'organisation :

```
{
  "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/*"
        }
      }
    }
  ]
}
```

Politiques relatives aux clusters multirégionaux

Pour les clusters multirégionaux, chaque cluster régional maintient sa propre politique de ressources, ce qui permet des contrôles spécifiques à la région. Voici un exemple de politiques différentes par région :

Politique us-east-1 :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

Politique us-east-2 :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> Les clés de contexte des conditions peuvent varier entre Régions AWS elles (par exemple, VPC IDs).

Blocage de l'accès public à l'aide de politiques basées sur les ressources dans Aurora DSQL

Le blocage de l'accès public (BPA) est une fonctionnalité qui identifie et empêche l'attachement de politiques basées sur les ressources qui accordent un accès public à vos clusters Aurora DSQL sur l'ensemble de vos comptes. AWS Avec le BPA, vous pouvez empêcher l'accès public à vos ressources SQL Aurora. Le BPA effectue des vérifications lors de la création ou de la modification d'une politique basée sur les ressources et contribue à améliorer votre niveau de sécurité avec Aurora DSQL.

BPA utilise un raisonnement automatisé pour analyser l'accès accordé par votre politique basée sur les ressources, et vous alerte si de telles autorisations sont détectées au moment de l'administration d'une politique basée sur les ressources. L'analyse vérifie l'accès à toutes les déclarations de politique basées sur les ressources, aux actions et à l'ensemble de clés de condition utilisées dans vos politiques.

> **Important**
>
> Le BPA contribue à protéger vos ressources en empêchant l'accès public par le biais de politiques basées sur les ressources directement associées à vos ressources Aurora DSQL,

telles que les clusters. En plus d'utiliser BPA, examinez attentivement les politiques suivantes pour vous assurer qu'elles n'accordent pas d'accès public :

- Politiques basées sur l'identité associées aux AWS principaux associés (par exemple, les rôles IAM)
- Politiques basées sur les AWS ressources associées (par exemple, AWS clés du service de gestion des clés (KMS))

Vous devez vous assurer que le principal n'inclut aucune entrée * ou que l'une des clés de condition spécifiées limite l'accès des principaux à la ressource. Si la politique basée sur les ressources accorde un accès public à votre cluster sur plusieurs AWS comptes, Aurora DSQL vous empêchera de créer ou de modifier la politique jusqu'à ce que la spécification contenue dans la stratégie soit corrigée et considérée comme non publique.

Vous pouvez rendre une politique non publique en spécifiant un ou plusieurs principaux à l'intérieur du bloc `Principal`. L'exemple de politique basée sur les ressources suivant bloque l'accès public en spécifiant deux principaux.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Les politiques qui restreignent l'accès en spécifiant certaines clés de condition ne sont pas non plus considérées comme publiques. Parallèlement à l'évaluation du principal spécifié dans la politique basée sur les ressources, les clés de condition fiables suivantes sont utilisées pour terminer l'évaluation d'une politique basée sur les ressources pour un accès non public :

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`

- aws:PrincipalOrgPaths
- aws:SourceAccount
- aws:SourceArn
- aws:SourceVpc
- aws:SourceVpce
- aws:UserId
- aws:PrincipalServiceName
- aws:PrincipalServiceNamesList
- aws:PrincipalIsAWSService
- aws:Ec2InstanceSourceVpc
- aws:SourceOrgID
- aws:SourceOrgPaths

En outre, pour qu'une politique basée sur les ressources ne soit pas publique, les valeurs d'Amazon Resource Name (ARN) et les clés de chaîne ne doivent pas contenir de caractères génériques ni de variables. Si votre politique basée sur les ressources utilise la clé `aws:PrincipalIsAWSService`, vous devez vous assurer que vous avez défini la valeur de la clé sur true.

La politique suivante limite l'accès à l'utilisateur `Ben` dans le compte spécifié. La condition impose une contrainte à `Principal` et ne doit pas être considérée comme publique.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

L'exemple suivant d'une politique basée sur des ressources non publique limite l'utilisation de `sourceVPC` avec l'opérateur `StringEquals`.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Opérations d'API SQL Aurora et politiques basées sur les ressources

Les politiques basées sur les ressources dans Aurora DSQL contrôlent l'accès à des opérations d'API spécifiques. Les sections suivantes répertorient toutes les opérations de l'API Aurora DSQL organisées par catégorie, avec une indication de celles qui prennent en charge les politiques basées sur les ressources.

La colonne Supports RBP indique si le fonctionnement de l'API est soumis à une évaluation de politique basée sur les ressources lorsqu'une politique est attachée au cluster.

Tag APIs

Opération API	Description	Supporte le RBP
ListTagsForResource	Répertorie les balises d'une ressource Aurora DSQL	Oui
TagResource	Ajoute des balises à une ressource SQL Aurora	Oui

Opération API	Description	Supporte le RBP
UntagResource	Supprime les balises d'une ressource SQL Aurora	Oui

Gestion des clusters APIs

Opération API	Description	Supporte le RBP
CreateCluster	Crée un nouveau cluster.	Non
DeleteCluster	Supprime un cluster	Oui
GetCluster	Récupère des informations sur un cluster	Oui
GetVpcEndpointServiceName	Récupère le nom du service de point de terminaison VPC pour un cluster	Oui
ListClusters	Répertorie les clusters de votre compte	Non
UpdateCluster	Met à jour la configuration d'un cluster	Oui

Propriété multirégionale APIs

Opération API	Description	Supporte le RBP
AddPeerCluster	Ajoute un cluster homologue à une configuration multirégionale	Oui
PutMultiRegionProperties	Définit les propriétés multirégionales d'un cluster	Oui
PutWitnessRegion	Définit la région témoin pour un cluster multirégional	Oui

Politique basée sur les ressources APIs

Opération API	Description	Supporte le RBP
DeleteClusterPolicy	Supprime la politique basée sur les ressources d'un cluster	Oui
GetClusterPolicy	Récupère la politique basée sur les ressources pour un cluster	Oui
PutClusterPolicy	Crée ou met à jour la politique basée sur les ressources pour un cluster	Oui

AWS Fault Injection Service APIs

Opération API	Description	Supporte le RBP
InjectError	Injecte des erreurs pour les tests d'injection de défauts	Non

Backup et restauration APIs

Opération API	Description	Supporte le RBP
GetBackupJob	Récupère les informations relatives à une tâche de sauvegarde	Non
GetRestoreJob	Récupère les informations relatives à une tâche de restauration	Non
StartBackupJob	Démarre une tâche de sauvegarde pour un cluster	Oui
StartRestoreJob	Démarre une tâche de restauration à partir d'une sauvegarde	Non
StopBackupJob	Arrête une tâche de sauvegarde en cours	Non

Opération API	Description	Supporte le RBP
StopRestoreJob	Arrête une tâche de restauration en cours	Non

Utilisation de rôles liés à un service pour Aurora DSQL

Aurora DSQL utilise des rôles liés à un service Gestion des identités et des accès AWS (IAM). Un rôle lié à un service est un type unique de rôle IAM lié directement à Aurora DSQL. Les rôles liés à un service sont prédéfinis par Aurora DSQL et incluent toutes les autorisations dont le service a besoin pour appeler au Services AWS nom de votre cluster Aurora DSQL.

Un rôle lié à un service simplifie la configuration des processus, car vous n'avez pas besoin d'ajouter manuellement les autorisations requises pour utiliser Aurora DSQL. Lorsque vous créez un cluster, Aurora DSQL crée automatiquement le rôle lié à un service pour vous. Vous pouvez supprimer le rôle lié à un service uniquement après avoir supprimé tous les clusters. Vos ressources Aurora DSQL sont ainsi protégées, car vous ne pouvez pas involontairement supprimer les autorisations nécessaires pour y accéder.

Pour plus d'informations sur les autres services qui prennent en charge les rôles liés à un service, reportez-vous aux Services AWS qui fonctionnent avec IAM et recherchez les services comportant un Oui dans la colonne Rôle lié à un service. Choisissez un Oui ayant un lien permettant de consulter les détails du rôle pour ce service.

Les rôles liés à un service sont disponibles dans toutes les régions Aurora DSQL prises en charge.

Autorisations des rôles liés à un service pour Aurora DSQL

Aurora DSQL utilise le rôle lié à un service nommé `AWSServiceRoleForAuroraDsql` — Permet à Amazon Aurora DSQL de créer et de gérer des AWS ressources en votre nom. Ce rôle lié à un service est attaché à la politique gérée suivante : AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Il se peut que vous rencontriez le message d'erreur suivant : `You don't have the permissions to create an Amazon Aurora DSQL service-linked role`. Si vous voyez ce message, vérifiez que vous avez activé les autorisations suivantes :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateDsqlServiceLinkedRole",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "dsql.amazonaws.com"
                }
            }
        }
    ]
}
```

Pour plus d'informations, consultez Autorisations des rôles liés à un service.

Créer un rôle lié à un service

Il n'est pas nécessaire de créer manuellement un rôle DSQLService LinkedRolePolicy lié à un service Aurora. Aurora DSQL crée automatiquement le rôle lié au service pour vous. Si le rôle DSQLService LinkedRolePolicy lié au service Aurora a été supprimé de votre compte, Aurora DSQL crée le rôle lorsque vous créez un nouveau cluster Aurora DSQL.

Modification d'un rôle lié à un service

Aurora DSQL ne vous permet pas de modifier le rôle lié au DSQLService LinkedRolePolicy service Aurora. Une fois que vous avez créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence au rôle. Vous pouvez toutefois modifier la description du rôle à l'aide de la console IAM, du AWS Command Line Interface (AWS CLI) ou de l'API IAM.

Supprimer un rôle lié à un service

Si vous n'avez plus besoin d'utiliser une fonction ou un service qui nécessite un rôle lié à un service, nous vous recommandons de supprimer ce rôle. De cette façon, vous n'avez aucune entité inutilisée qui n'est pas surveillée ou gérée activement.

Avant de pouvoir supprimer un rôle lié à un service pour un compte, vous devez supprimer tous les clusters du compte.

Vous pouvez utiliser la console IAM AWS CLI, ou l'API IAM pour supprimer un rôle lié à un service. Pour plus d'informations, consultez Création d'un rôle lié à un service dans le Guide de l'utilisateur IAM.

Régions prises en charge pour les rôles liés à un service Aurora DSQL

Aurora DSQL prend en charge l'utilisation des rôles liés à un service dans toutes les régions où le service est disponible. Pour plus d'informations, consultez Régions et points de terminaison AWS.

Utilisation des clés de condition IAM avec Amazon Aurora DSQL

Lorsque vous accordez des autorisations dans Aurora DSQL, vous pouvez spécifier des conditions pour déterminer comment une politique d'autorisation doit prendre effet. Les exemples suivants montrent comment vous pouvez utiliser des clés de condition dans les politiques d'autorisation Aurora DSQL.

Exemple 1 : accorder l'autorisation de créer un cluster dans un environnement spécifique Région AWS

La politique suivante accorde l'autorisation de créer des clusters dans les régions USA Est (Virginie du Nord) et USA Est (Ohio). Cette politique utilise l'ARN de la ressource pour limiter les régions autorisées. Aurora DSQL ne peut donc créer des clusters que si cet ARN est spécifié dans la section `Resource` de la politique.

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"
        }
    ]
}
```

Exemple 2 : Accorder l'autorisation de créer un cluster multirégional dans des domaines spécifiques Région AWS

La politique suivante accorde l'autorisation de créer des clusters multi-régions dans les régions USA Est (Virginie du Nord) et USA Est (Ohio). Cette politique utilise l'ARN de la ressource pour limiter les régions autorisées. Aurora DSQL ne peut donc créer des clusters multi-régions que si cet ARN est spécifié dans la section `Resource` de la politique. Notez que la création de clusters multi-régions nécessite également les autorisations `PutMultiRegionProperties`, `PutWitnessRegion` et `AddPeerCluster` dans chaque région spécifiée.

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/*"
        ]
      }
    ]
```

```
}
```

Exemple 3 : accorder l'autorisation de créer un cluster multi-régions dans une région témoin spécifique

La politique suivante utilise une clé de condition `dsql:WitnessRegion` Aurora DSQL et permet à un utilisateur de créer des clusters multi-régions avec une région témoin dans la région USA Ouest (Oregon). Si vous ne spécifiez pas la condition `dsql:WitnessRegion`, vous pouvez utiliser n'importe quelle région comme région témoin.

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"
                    ]
                }
            }
        }
    ]
}
```

Réponse aux incidents dans Amazon Aurora DSQL

Chez AWS, la sécurité est la priorité numéro 1. Dans le cadre du modèle de responsabilité partagée du AWS cloud, AWS gère un centre de données, un réseau et une architecture logicielle qui répondent aux exigences des organisations les plus sensibles en matière de sécurité. AWS est responsable de toute réponse aux incidents concernant le service Amazon Aurora DSQL lui-même. De plus, en tant que AWS client, vous partagez la responsabilité du maintien de la sécurité dans le cloud. Cela signifie que vous contrôlez la sécurité que vous choisissez de mettre en œuvre à partir des AWS outils et des fonctionnalités auxquels vous avez accès. En outre, vous êtes responsable de la réponse aux incidents dans le cadre du modèle de responsabilité partagée.

En établissant une base de sécurité répondant aux objectifs de vos applications exécutées dans le cloud, vous êtes en mesure de détecter les écarts auxquels vous pouvez réagir. Pour vous aider à comprendre l'impact de la réponse aux incidents et de vos choix sur les objectifs de votre entreprise, nous vous encourageons à consulter les ressources suivantes :

- AWS Guide de réponse aux incidents de sécurité
- AWS Meilleures pratiques en matière de sécurité, d'identité et de conformité
- Livre blanc sur la perspective de sécurité du cadre d'adoption du AWS cloud (CAF)

Amazon GuardDuty est un service géré de détection des menaces qui surveille en permanence les comportements malveillants ou non autorisés afin d'aider les clients à protéger Comptes AWS leur charge de travail et à identifier les activités suspectes potentielles avant qu'elles ne dégénèrent en incident. Il surveille les activités telles que les appels d'API inhabituels ou les déploiements potentiellement non autorisés indiquant une possible compromission du compte ou des ressources ou une reconnaissance par des acteurs malveillants. Par exemple, Amazon GuardDuty est en mesure de détecter des activités suspectes dans Amazon Aurora DSQL APIs, comme la connexion d'un utilisateur depuis un nouvel emplacement et la création d'un nouveau cluster.

Validation de la conformité pour Amazon Aurora DSQL

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Résilience dans Amazon Aurora DSQL

L'infrastructure AWS mondiale est construite autour Régions AWS de zones de disponibilité (AZ). Régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données. Aurora DSQL est conçu pour que vous puissiez tirer parti de l'infrastructure AWS régionale tout en fournissant la meilleure disponibilité de base de données. Par défaut, les clusters à une seule région d'Aurora DSQL offrent une disponibilité multi-AZ, ce qui permet de tolérer les défaillances majeures des composants et les perturbations de l'infrastructure susceptibles d'avoir un impact sur l'accès à une zone de disponibilité complète. Les clusters multi-régions offrent tous les avantages de la résilience multi-AZ tout en garantissant une disponibilité des bases de données très constante, même dans les cas où la Région AWS n'est pas accessible aux clients de l'application.

Pour plus d'informations sur les zones de disponibilité Régions AWS et les zones de disponibilité, consultez la section Infrastructure AWS globale.

Outre l'infrastructure AWS globale, Aurora DSQL propose plusieurs fonctionnalités pour répondre à vos besoins en matière de résilience et de sauvegarde des données.

Sauvegarde et restauration

Aurora DSQL prend en charge la sauvegarde et la restauration avec Console AWS Backup. Vous pouvez effectuer une sauvegarde et une restauration complètes pour vos clusters à une seule région ou multi-régions. Pour de plus amples informations, veuillez consulter Sauvegarde et restauration pour Amazon Aurora DSQL.

Réplication

De par sa conception, Aurora DSQL valide toutes les transactions d'écriture dans un journal de transactions distribué et réplique de manière synchrone toutes les données du journal validées dans des répliques de stockage utilisateur en trois exemplaires. AZs Les clusters multi-régions fournissent des fonctionnalités complètes de réplication entre plusieurs régions entre les régions de lecture et d’écriture.

Une région témoin désignée prend en charge les écritures dans le journal des transactions uniquement et ne consomme pas d’espace de stockage. Les régions témoins ne disposent pas d’un point de terminaison. Cela signifie que les régions témoins ne stockent que des journaux de transactions chiffrés, ne nécessitent aucune administration ni configuration et ne sont pas accessibles aux utilisateurs. Si la région témoin est altérée, cela n'a aucun impact sur la disponibilité du cluster. Les transactions d'écriture peuvent connaître une légère augmentation de la latence jusqu'à ce que la région témoin se rétablisse.

Les journaux de transactions et le stockage utilisateur Aurora DSQL sont distribués avec toutes les données présentées aux processeurs de requêtes Aurora DSQL sous la forme d’un volume logique unique. Aurora DSQL divise, fusionne et réplique automatiquement les données en fonction de la plage de clés primaires de la base de données et des modèles d’accès. Aurora DSQL met à l’échelle automatiquement les réplicas en lecture, à la hausse comme à la baisse, en fonction de la fréquence d’accès en lecture.

Les réplicas de stockage en cluster sont répartis sur une flotte de stockage à locataires multiples. Si un composant ou une AZ est endommagé, Aurora DSQL redirige automatiquement l’accès aux composants survivants et répare de manière asynchrone les réplicas manquants. Une fois qu’Aurora DSQL a corrigé les réplicas défectueux, Aurora DSQL les ajoute automatiquement au quorum de stockage et les met à la disposition de votre cluster.

Haute disponibilité

Par défaut, les clusters à une seule région et multi-régions dans Aurora DSQL sont actifs-actifs, et il n’est pas nécessaire de provisionner, configurer ni reconfigurer manuellement des clusters. Aurora DSQL automatise entièrement la restauration des clusters, ce qui élimine le besoin d’opérations de basculement principales-secondaires traditionnelles. La réplication est toujours synchrone et effectuée en plusieurs AZs exemplaires. Il n'y a donc aucun risque de perte de données en cas de retard de réplication ou de basculement vers une base de données secondaire asynchrone en cas de reprise après échec.

Les clusters à région unique fournissent un point de terminaison redondant multi-AZ qui permet automatiquement un accès simultané avec une forte cohérence des données entre les trois. AZs Cela signifie que les répliques de stockage utilisateur sur l'un de ces trois AZs types renvoient toujours le même résultat à un ou plusieurs lecteurs et sont toujours disponibles pour recevoir des écritures. Cette forte cohérence et cette résilience multi-AZ sont disponibles dans toutes les régions pour les clusters multi-régions Aurora DSQL. Cela signifie que les clusters multi-régions fournissent deux points de terminaison régionaux très cohérents, de sorte que les clients peuvent lire ou écrire sans distinction dans l'une ou l'autre région sans aucun délai de réplication lors de la validation.

Aurora DSQL assure une disponibilité de 99,99 % pour les clusters à une seule région et de 99,999 % pour les clusters multi-régions.

Test d'injection de pannes

Amazon Aurora DSQL s'intègre à AWS Fault Injection Service (AWS FIS), un service entièrement géré permettant d'exécuter des expériences d'injection contrôlée de défauts afin d'améliorer la résilience d'une application. En utilisant AWS FIS, vous pouvez :

- Créer des modèles d'expérimentation qui définissent des scénarios de pannes spécifiques
- Injecter les pannes (taux d'erreur de connexion au cluster élevés) pour valider les mécanismes de gestion des erreurs et de restauration des applications
- Testez le comportement des applications multirégionales pour valider le transfert du trafic des applications entre les Régions AWS périodes où le taux d'erreur de connexion Région AWS est élevé

Par exemple, dans un cluster multi-régions couvrant les régions USA Est (Virginie du Nord) et USA Est (Ohio), vous pouvez exécuter une expérience dans la région USA Est (Ohio) pour y tester les pannes pendant que la région USA Est (Virginie du Nord) poursuit ses activités normales. Ces tests contrôlés vous aident à identifier et à résoudre les problèmes potentiels avant qu'ils n'affectent les charges de travail de production.

Consultez la section Objectifs d'action dans le guide de AWS FIS l'utilisateur pour obtenir la liste complète des actions AWS FIS prises en charge.

Pour plus d'informations sur les actions Amazon Aurora DSQL disponibles dans AWS FIS, consultez la référence des actions Aurora DSQL dans le guide de l'AWS FIS utilisateur.

Pour commencer à exécuter des expériences d'injection de pannes, consultez Planification de vos expériences AWS FIS dans le guide de l'utilisateur AWS FIS .

Sécurité de l'infrastructure dans Amazon Aurora DSQL

En tant que service géré, Amazon Aurora DSQL est protégé par les procédures de sécurité du réseau AWS mondial décrites dans la section Meilleures pratiques en matière de sécurité, d'identité et de conformité.

Vous utilisez des appels d'API AWS publiés pour accéder à Aurora DSQL via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.2 ou version ultérieure. Les clients doivent aussi prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Gestion et connexion aux clusters SQL Amazon Aurora à l'aide de AWS PrivateLink

Avec AWS PrivateLink Amazon Aurora DSQL, vous pouvez configurer des points de terminaison Amazon VPC (points de terminaison d'interface) dans votre Amazon Virtual Private Cloud. Ces points de terminaison sont directement accessibles depuis des applications installées sur site via Amazon VPC et/ou via un Direct Connect autre système de peering Région AWS via Amazon VPC. En utilisant AWS PrivateLink et en interfacant les points de terminaison, vous pouvez simplifier la connectivité réseau privé entre vos applications et Aurora DSQL.

Les applications de votre Amazon VPC peuvent accéder à Aurora DSQL via les points de terminaison de l'interface Amazon VPC sans avoir besoin d'adresses IP publiques.

Les points de terminaison d'interface sont représentés par une ou plusieurs interfaces réseau élastiques (ENIs) auxquelles des adresses IP privées sont attribuées à partir de sous-réseaux de votre Amazon VPC. Les demandes adressées à Aurora DSQL via les points de terminaison de l'interface restent sur le AWS réseau. Pour plus d'informations sur la façon de connecter votre Amazon VPC à votre réseau sur site, consultez le Guide de l'utilisateur Direct Connect et le Guide de l'utilisateur VPN AWS Site-to-Site VPN.

Pour des informations générales sur les points de terminaison d'interface, consultez la section Accès à un AWS service à l'aide d'un point de terminaison Amazon VPC d'interface dans AWS PrivateLinkle guide de l'utilisateur.

Types de points de terminaison Amazon VPC pour Aurora DSQL

Aurora DSQL nécessite deux types de points de AWS PrivateLink terminaison différents.

1. Point de terminaison de gestion : ce point de terminaison est utilisé pour les opérations administratives telles que `get`, `create`, `update`, `delete` et `list` sur les clusters Aurora DSQL. Consultez Gestion des clusters SQL Aurora à l'aide de AWS PrivateLink.
2. Point de terminaison de connexion : ce point de terminaison est utilisé pour la connexion aux clusters Aurora DSQL via les clients PostgreSQL. Consultez Connexion aux clusters SQL Aurora à l'aide de AWS PrivateLink.

Considérations relatives à l'utilisation AWS PrivateLink d'Aurora DSQL

Les considérations relatives à Amazon VPC s'appliquent à AWS PrivateLink Aurora DSQL. Pour plus d'informations, consultez la section Accès à un AWS service à l'aide d'un point de terminaison VPC d'interface et de AWS PrivateLink quotas dans le AWS PrivateLink Guide.

Gestion des clusters SQL Aurora à l'aide de AWS PrivateLink

Vous pouvez utiliser le AWS Command Line Interface ou les kits de développement AWS logiciel (SDKs) pour gérer les clusters Aurora DSQL via les points de terminaison de l'interface Aurora DSQL.

Création d'un point de terminaison Amazon VPC

Pour créer un point de terminaison d'interface Amazon VPC, consultez la section Créer un point de terminaison Amazon VPC dans le Guide. AWS PrivateLink

```
aws ec2 create-vpc-endpoint \
--region region \
--service-name com.amazonaws.region.dsql \
--vpc-id your-vpc-id \
--subnet-ids your-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id \
```

Pour utiliser le nom DNS régional par défaut pour les demandes d'API Aurora DSQL, ne désactivez pas le DNS privé lorsque vous créez le point de terminaison de l'interface Aurora DSQL. Lorsque le DNS privé est activé, les demandes adressées au service Aurora DSQL depuis votre Amazon VPC sont automatiquement résolues vers l'adresse IP privée du point de terminaison Amazon VPC, plutôt que vers le nom DNS public. Lorsque le DNS privé est activé, les demandes Aurora DSQL effectuées au sein de votre Amazon VPC sont automatiquement résolues vers votre point de terminaison Amazon VPC.

Si le DNS privé n'est pas activé, utilisez les `--endpoint-url` paramètres `--region` et avec les AWS CLI commandes pour gérer les clusters Aurora DSQL via les points de terminaison de l'interface Aurora DSQL.

Établissement de la liste des clusters à l'aide d'une URL de point de terminaison

Dans l'exemple suivant, remplacez le nom DNS Région AWS `us-east-1` et le nom DNS de l'ID du point de terminaison Amazon VPC `vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com` par vos propres informations.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com list-clusters
```

Opérations d'API

Reportez-vous à la référence de l'API Aurora DSQL pour obtenir de la documentation sur la gestion des ressources dans Aurora DSQL.

Gestion des politiques de point de terminaison

En testant et en configurant de manière approfondie les politiques relatives aux points de terminaison Amazon VPC, vous pouvez garantir que votre cluster Aurora DSQL est sécurisé et conforme aux exigences de gouvernance et de contrôle d'accès spécifiques de votre organisation.

Exemple : stratégie d'accès complète à Aurora DSQL

La stratégie suivante accorde l'accès total à toutes les actions et ressources Aurora DSQL via le point de terminaison Amazon VPC spécifié.

```
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": "*"
        }
      ]
    }'
```

Exemple : stratégie d'accès restreinte à Aurora DSQL

La stratégie suivante autorise uniquement ces actions Aurora DSQL.

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

Toutes les autres actions Aurora DSQL sont refusées.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Connexion aux clusters SQL Aurora à l'aide de AWS PrivateLink

Une fois que votre AWS PrivateLink point de terminaison est configuré et actif, vous pouvez vous connecter à votre cluster Aurora DSQL à l'aide d'un client PostgreSQL. Les instructions de connexion ci-dessous décrivent les étapes à suivre pour créer le nom d'hôte approprié pour la connexion via le AWS PrivateLink point de terminaison.

Configuration d'un point de terminaison de AWS PrivateLink connexion

Étape 1 : obtenir le nom du service pour votre cluster

Lorsque vous créez un AWS PrivateLink point de terminaison pour vous connecter à votre cluster, vous devez d'abord récupérer le nom du service spécifique au cluster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Exemple de réponse

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

Le nom du service inclut un identifiant, comme `dsql-fnh4` dans l'exemple. Cet identifiant est également nécessaire lors de la construction du nom d'hôte pour la connexion à votre cluster.

AWS SDK for Python (Boto3)

```
import boto3

dsql_client = boto3.client('dsql', region_name='us-east-1')
response = dsql_client.get_vpc_endpoint_service_name(
    identifier='your-cluster-id'
)
service_name = response['serviceName']
print(f"Service Name: {service_name}")
```

AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameRequest;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameResponse;

String region = "us-east-1";
String clusterId = "your-cluster-id";

DsqlClient dsqlClient = DsqlClient.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

GetVpcEndpointServiceNameResponse response = dsqlClient.getVpcEndpointServiceName(
    GetVpcEndpointServiceNameRequest.builder()
        .identifier(clusterId)
        .build()
);
String serviceName = response.serviceName();
System.out.println("Service Name: " + serviceName);
```

Étape 2 : créer le point de terminaison Amazon VPC

À l'aide du nom de service obtenu à l'étape précédente, créez un point de terminaison Amazon VPC.

> **Important**
>
> Les instructions de connexion ci-dessous ne fonctionnent que pour la connexion aux clusters lorsque le mode privé est activé par le DNS. N'utilisez pas l'indicateur `--no-private-dns-enabled` lors de la création du point de terminaison, car cela empêcherait les instructions de connexion ci-dessous de fonctionner correctement. Si vous désactivez le DNS privé, vous devrez créer votre propre enregistrement DNS privé joker pointant vers le point de terminaison créé.

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
```

Exemple de réponse

```
{
    "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

Utiliser une URL de point de terminaison pour 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);
```

Configuration supplémentaire lors de la connexion Direct Connect ou du peering Amazon VPC

Une configuration supplémentaire peut être nécessaire pour se connecter aux clusters Aurora DSQL à l'aide d'un point de terminaison de AWS PrivateLink connexion provenant d'appareils sur site via Amazon VPC peering ou. Direct Connect Cette configuration n'est pas requise si votre application s'exécute dans le même Amazon VPC que votre AWS PrivateLink point de terminaison. Les entrées DNS privées créées ci-dessus ne seront pas résolues correctement en dehors du VPC Amazon du point de terminaison, mais vous pouvez créer vos propres enregistrements DNS privés qui sont résolus vers votre point de terminaison de AWS PrivateLink connexion.

Créez un enregistrement DNS CNAME privé qui pointe vers le nom de domaine complet du AWS PrivateLink point de terminaison. Le nom de domaine de l'enregistrement DNS créé doit être construit à partir des composants suivants :

1. L'identifiant du service issu du nom du service. Par exemple : `dsql-fnh4`
2. Le Région AWS

Créez l'enregistrement DNS CNAME avec un nom de domaine au format suivant : `*.`*`service-identifier`*`.`*`region`*`.on.aws`

Le format du nom de domaine est important pour deux raisons :

1. Le nom d'hôte utilisé pour se connecter à Aurora DSQL doit correspondre au certificat de serveur d'Aurora DSQL lorsque vous utilisez le `verify-full` mode SSL. Cela garantit le plus haut niveau de sécurité de connexion.
2. Aurora DSQL utilise la partie ID de cluster du nom d'hôte utilisé pour se connecter à Aurora DSQL afin d'identifier le cluster qui se connecte.

S'il n'est pas possible de créer des enregistrements DNS privés, vous pouvez toujours vous connecter à Aurora DSQL. Consultez Connexion à un cluster Aurora DSQL à l'aide d'un AWS PrivateLink point de terminaison sans DNS privé.

Connexion à un cluster SQL Aurora à l'aide d'un point de terminaison de AWS PrivateLink connexion

Une fois que votre AWS PrivateLink point de terminaison est configuré et actif (vérifiez qu'il l'`Stateestavailable`), vous pouvez vous connecter à votre cluster Aurora DSQL à l'aide d'un client PostgreSQL. Pour obtenir des instructions sur l'utilisation de AWS SDKs, vous pouvez suivre les guides de la section Programmation avec Aurora DSQL. Vous devez modifier le point de terminaison du cluster pour qu'il corresponde au format du nom d'hôte.

Construction du nom d'hôte

Le nom d'hôte pour la connexion est AWS PrivateLink différent du nom d'hôte DNS public. Vous devez le construire à l'aide des composants suivants.

1. `Your-cluster-id`
2. L'identifiant du service issu du nom du service. Par exemple : `dsql-fnh4`
3. Le Région AWS. Par exemple : `us-east-1`

Utilisez le format suivant : *`cluster-id`*`.`*`service-identifier`*`.`*`region`*`.on.aws`

Exemple : connexion à l'aide de 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
```

Connexion à un cluster Aurora DSQL à l'aide d'un AWS PrivateLink point de terminaison sans DNS privé

Les instructions de connexion ci-dessus s'appuient sur des enregistrements DNS privés. Si votre application s'exécute dans le même Amazon VPC que votre AWS PrivateLink point de terminaison, les enregistrements DNS sont créés pour vous. Si vous vous connectez à partir d'appareils sur site via Amazon VPC peering Direct Connect, vous pouvez également créer vos propres enregistrements DNS privés. Cependant, la configuration des enregistrements DNS n'est pas toujours possible en raison des restrictions réseau imposées par vos équipes de sécurité. Si votre application doit se connecter via Direct Connect ou depuis un Amazon VPC homologue et que la configuration d'un enregistrement DNS n'est pas possible, vous pouvez toujours vous connecter à Aurora DSQL.

Aurora DSQL utilise la partie ID de cluster de votre nom d'hôte pour identifier le cluster qui se connecte, mais si la configuration d'un enregistrement DNS n'est pas possible, Aurora DSQL prend en charge la spécification du cluster cible à l'aide de l'`amzn-cluster-id`option de connexion. Avec cette option, il est possible d'utiliser le nom de domaine complet de votre AWS PrivateLink terminal comme nom d'hôte lors de la connexion.

> **Important**
>
> Lorsque vous vous connectez avec le nom de domaine complet ou l'adresse IP de votre AWS PrivateLink terminal, le mode `verify-full` SSL n'est pas pris en charge. C'est pourquoi il est préférable de configurer un DNS privé.

Exemple : Spécification de l'option de connexion avec l'ID du cluster à l'aide de 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
```

Résolution des problèmes liés à AWS PrivateLink

Problèmes courants et solutions correspondantes

Le tableau suivant répertorie les problèmes courants et les solutions liés à AWS PrivateLink avec Aurora DSQL.

Problème	Cause possible	Solution
Délai de connexion	Le groupe de sécurité n'est pas correctem ent configuré	Utilisez l'analyseur d'accessibilité Amazon VPC pour vous assurer que votre configuration réseau autorise le trafic sur le port 5432.
Échec de résolution DNS	DNS privé non activé	Vérifiez que le point de terminaison Amazon VPC a été créé avec le DNS privé activé.
Échec d'authent ification	Informations d'identif ication incorrectes ou jeton expiré	Générez un nouveau jeton d'authentification et vérifiez le nom d'utilisateur.
Nom de service introuvable	ID de cluster incorrect	Vérifiez l'ID de votre cluster et vérifiez le nom du service Région AWS lorsque vous récupérez le nom du service.

Ressources connexes

Pour plus d'informations, consultez les ressources suivantes :

- Guide de l'utilisateur Amazon Aurora DSQL
- Documentation AWS PrivateLink
- Accédez aux AWS services via AWS PrivateLink

Configuration et analyse des vulnérabilités dans Amazon Aurora DSQL

AWS gère les tâches de sécurité de base telles que l'application de correctifs au système d'exploitation client (OS) et aux bases de données, la configuration du pare-feu et la reprise après sinistre. Ces procédures ont été vérifiées et certifiées par les tiers appropriés. Pour plus de détails, consultez les ressources suivantes :

- Modèle de responsabilité partagée
- Amazon Web Services : présentation des procédures de sécurité (livre blanc)

Prévention du cas de figure de l'adjoint désorienté entre services

Le problème de député confus est un problème de sécurité dans lequel une entité qui n'est pas autorisée à effectuer une action peut contraindre une entité plus privilégiée à le faire. En AWS, l'usurpation d'identité interservices peut entraîner un problème de confusion chez les adjoints. L'usurpation d'identité entre services peut se produire lorsqu'un service (le service appelant) appelle un autre service (le service appelé). Le service appelant peut être manipulé et ses autorisations utilisées pour agir sur les ressources d'un autre client auxquelles on ne serait pas autorisé à accéder autrement. Pour éviter cela, AWS fournit des outils qui vous aident à protéger vos données pour tous les services avec des principaux de service qui ont eu accès aux ressources de votre compte.

Nous vous recommandons d'utiliser les clés de contexte de condition globale `aws:SourceArn` et `aws:SourceAccount` dans les politiques de ressources afin de limiter les autorisations à la ressource octroyées par Amazon Aurora DSQL à un autre service. Utilisez `aws:SourceArn` si vous souhaitez qu'une seule ressource soit associée à l'accès entre services. Utilisez `aws:SourceAccount` si vous souhaitez autoriser l'association d'une ressource de ce compte à l'utilisation interservices.

Le moyen le plus efficace de se protéger contre le problème de député confus consiste à utiliser la clé de contexte de condition globale `aws:SourceArn` avec l'ARN complet de la ressource. Si vous ne connaissez pas l'ARN complet de la ressource ou si vous spécifiez plusieurs ressources, utilisez la clé de contexte de condition globale `aws:SourceArn` avec des caractères génériques (*) pour les parties inconnues de l'ARN. Par exemple, `arn:aws:dsql:*:123456789012:*`.

Si la valeur `aws:SourceArn` ne contient pas l'ID du compte, tel qu'un ARN de compartiment Amazon S3, vous devez utiliser les deux clés de contexte de condition globale pour limiter les autorisations.

La valeur de `aws:SourceArn` doit être ResourceDescription.

L'exemple suivant montre comment utiliser les clés de contexte de condition globale `aws:SourceArn` et `aws:SourceAccount` dans Aurora DSQL afin d'éviter le problème de l'adjoint confus.

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"
      }
    }
  }
}
```

Bonnes pratiques de sécurité pour Aurora DSQL

Aurora DSQL fournit différentes fonctionnalités de sécurité à prendre en compte lorsque vous développez et implémentez vos propres politiques de sécurité. Les bonnes pratiques suivantes doivent être considérées comme des instructions générales et ne représentent pas une solution de sécurité complète. Étant donné que ces bonnes pratiques peuvent ne pas être appropriées ou suffisantes pour votre environnement, considérez-les comme des remarques utiles plutôt que comme des recommandations.

Rubriques

- Bonnes pratiques de sécurité de détection pour Aurora DSQL
- Bonnes pratiques de sécurité préventive pour Aurora DSQL

Bonnes pratiques de sécurité de détection pour Aurora DSQL

Outre les méthodes suivantes pour utiliser Aurora DSQL en toute sécurité, consultez Security in AWS Well-Architected Tool pour découvrir comment les technologies cloud améliorent votre sécurité.

CloudWatch Alarmes Amazon

À l'aide des CloudWatch alarmes Amazon, vous observez une seule métrique sur une période que vous spécifiez. Si la métrique dépasse un seuil donné, une notification est envoyée à une rubrique ou AWS Auto Scaling à une politique Amazon SNS. CloudWatch les alarmes n'appellent pas d'actions car elles se trouvent dans un état particulier. L'état doit avoir changé et avoir été conservé pendant un nombre de périodes spécifié.

Étiquetage de vos ressources Aurora DSQL pour l'identification et l'automatisation

Vous pouvez attribuer des métadonnées à vos AWS ressources sous forme de balises. Chaque étiquette est un libellé composé d'une clé définie par le client et d'une valeur facultative qui peut faciliter la gestion, la recherche et le filtrage de ressources.

L'étiquetage permet l'implémentation de contrôles groupés. Bien qu'il n'existe pas de types intrinsèques d'étiquettes, celles-ci vous permettent de catégoriser des ressources par objectif, par propriétaire, par environnement ou selon d'autres critères. Voici quelques exemples :

- Sécurité : utilisée pour déterminer des exigences telles que le chiffrement.
- Confidentialité – Identifiant pour le niveau spécifique de confidentialité des données qu'une ressource prend en charge.

- Environnement : utilisé pour différencier les infrastructures de développement, de test et de production.

Vous pouvez attribuer des métadonnées à vos AWS ressources sous forme de balises. Chaque étiquette est un libellé composé d'une clé définie par le client et d'une valeur facultative qui peut faciliter la gestion, la recherche et le filtrage de ressources.

L'étiquetage permet l'implémentation de contrôles groupés. Bien qu'il n'existe pas de types de balises inhérents, elles vous permettent de classer les ressources par objectif, par propriétaire, par environnement ou selon d'autres critères. Voici quelques exemples.

- Sécurité : utilisée pour déterminer des exigences telles que le chiffrement.
- Confidentialité : identifiant pour le niveau spécifique de confidentialité des données qu'une ressource prend en charge.
- Environnement : utilisé pour différencier les infrastructures de développement, de test et de production.

Pour plus d'informations, consultez la section Meilleures pratiques en matière de balisage AWS des ressources.

Bonnes pratiques de sécurité préventive pour Aurora DSQL

Outre les méthodes suivantes pour utiliser Aurora DSQL en toute sécurité, consultez Security in AWS Well-Architected Tool pour découvrir comment les technologies cloud améliorent votre sécurité.

Utilisation des rôles IAM pour authentifier l'accès à Aurora DSQL.

Les utilisateurs, applications et autres personnes Services AWS qui accèdent à Aurora DSQL doivent inclure des AWS informations d'identification valides dans AWS l'API et les AWS CLI demandes. Vous ne devez pas stocker les AWS informations d'identification directement dans l'application ou dans les instances EC2. Il s'agit d'informations d'identification à long terme qui ne font pas l'objet d'une rotation automatique. La compromission de ces informations d'identification a un impact commercial significatif. Un rôle IAM vous permet d'obtenir des clés d'accès temporaires que vous pouvez utiliser pour accéder Services AWS aux ressources.

Pour de plus amples informations, veuillez consulter Authentification et autorisation pour Aurora DSQL.

Utilisation des politiques IAM pour l'autorisation de base Aurora DSQL.

Lorsque vous accordez des autorisations, vous décidez qui les reçoit, les opérations d'API Aurora DSQL auxquelles elles s'appliquent, et les actions spécifiques que vous souhaitez autoriser sur ces ressources. L'implémentation d'un privilège minimum est la clé de la réduction des risques de sécurité et de l'impact potentiel d'erreurs ou d'actes de malveillance.

Associez des politiques d'autorisation aux rôles IAM et accordez des autorisations pour exécuter des opérations sur des ressources Aurora DSQL. Des limites d'autorisations pour les entités IAM sont également disponibles, ce qui vous permet de définir les autorisations maximales qu'une politique basée sur l'identité peut accorder à une entité IAM.

À l'instar des meilleures pratiques relatives à l'utilisateur root Compte AWS, n'utilisez pas le `admin` rôle dans Aurora DSQL pour effectuer des opérations quotidiennes. Nous vous recommandons plutôt de créer des rôles de base de données personnalisés pour gérer votre cluster et vous y connecter. Pour plus d'informations, consultez Accès à Aurora DSQL et Comprendre l'authentification et l'autorisation pour Aurora DSQL.

Utilisation de **`verify-full`** dans les environnements de production.

Ce paramètre vérifie que le certificat du serveur est signé par une autorité de certification fiable et que le nom d'hôte du serveur correspond au certificat.

Mise à jour de votre client PostgreSQL

Mettez régulièrement à jour votre client PostgreSQL vers la dernière version pour bénéficier des améliorations de sécurité. Nous vous recommandons d'utiliser la version 17 de PostgreSQL.

Balisage de ressources dans Aurora DSQL

Dans AWS, les balises sont des paires clé/valeur définies par l'utilisateur que vous définissez et associez aux ressources Aurora DSQL telles que les clusters. Les balises sont facultatives. Si vous fournissez une clé, la valeur est facultative.

Vous pouvez utiliser la AWS Management Console, l'AWS CLI ou les kits AWS SDK pour ajouter, répertorier et supprimer des balises sur les clusters Aurora DSQL. Vous pouvez ajouter des balises pendant et après la création du cluster à l'aide de la console AWS. Pour attribuer des balises à un cluster après sa création avec l'AWS CLI, utilisez l'opération `TagResource`.

Balisage des clusters avec un nom

Aurora DSQL crée des clusters avec un identifiant unique à l'échelle mondiale, appelé Amazon Resource Name (ARN). Si vous souhaitez attribuer un nom convivial à votre cluster, nous vous recommandons d'utiliser une balise.

Si vous créez une console avec la console Aurora DSQL, Aurora DSQL crée automatiquement une balise. Cette balise a pour clé Nom et une valeur générée automatiquement qui représente le nom du cluster. Cette valeur est configurable afin que vous puissiez attribuer un nom plus convivial à votre cluster. Si un cluster possède une balise Nom et une valeur associée, vous pouvez voir la valeur dans toute la console Aurora DSQL.

Balisage des exigences

Les balises possèdent les exigences suivantes :

- Les clés ne peuvent pas être préfixées par `aws:`.
- Les clés doivent être uniques par ensemble de balises.
- Une clé doit comporter entre 1 et 128 caractères autorisés.
- Une valeur doit comprendre entre 0 et 256 caractères autorisés.
- Les valeurs ne doivent pas être uniques par ensemble de balises.
- Les caractères autorisés pour les clés et les valeurs sont les lettres, les chiffres, les espaces et les symboles suivants : _ . : / = + - @.
- Les clés et les valeurs sont sensibles à la casse.

Balise des notes d'utilisation

Lorsque vous utilisez des balises dans Aurora DSQL, tenez compte des éléments suivants.

- Lorsque vous utilisez l'AWS CLI ou l'API Aurora DSQL, assurez-vous de fournir l'Amazon Resource Name (ARN) pour la ressource Aurora DSQL avec laquelle vous souhaitez travailler. Pour plus d'informations, consultez Format de l'Amazon Resource Name (ARN) pour les ressources Aurora DSQL.
- Chaque ressource possède un ensemble de balises, lequel constitue un ensemble d'une ou de plusieurs balises affectées à la ressource.
- Chaque ressource peut avoir jusqu'à 50 balises par ensemble de balises.
- Si vous supprimez une ressource, les balises associées sont supprimées.
- Vous pouvez ajouter des balises lorsque vous créez une ressource, vous pouvez afficher et modifier des balises à l'aide des opérations d'API suivantes : `TagResource`, `UntagResource` et `ListTagsForResource`.
- Vous pouvez utiliser des balises avec des politiques IAM. Vous pouvez les utiliser pour gérer l'accès aux cluster Aurora DSQL et contrôler les actions qui peuvent être appliquées aux ressources. Pour plus d'informations, consultez Contrôle de l'accès aux ressources AWS.
- Vous pouvez utiliser des balises pour diverses autres activités sur AWS. Pour plus d'informations, consultez Stratégies de balisage courantes.

Considérations relatives à l'utilisation d Amazon Aurora DSQL

Lorsque vous travaillez avec Amazon Aurora DSQL, tenez compte des facteurs suivants. Pour plus d'informations sur la compatibilité PostgreSQL, consultez Compatibilité des fonctionnalités SQL dans Aurora DSQL. Pour les quotas et les limites, consultez Quotas de cluster et limites de base de données dans Amazon Aurora DSQL.

- Le calcul de la limite de stockage peut prendre un certain temps pour refléter le stockage libéré après l'exécution d'une `DROP TABLE` commande. Si vous avez besoin d'une capacité de stockage supplémentaire, consultez la section Quotas de cluster pour demander des mises à jour des quotas.
- Pour les grandes tables dans Aurora DSQL, utilisez le catalogue système pour récupérer le nombre de lignes des tables plutôt que les `COUNT(*)` opérations. Pour plus d'informations, consultez la section Utilisation des tables et des commandes systèmes dans Aurora DSQL.
- Aurora DSQL gère les autorisations par le biais d'autorisations au niveau du schéma. Les utilisateurs administrateurs créent des schémas en utilisant `CREATE SCHEMA` et accordent l'accès à d'autres rôles à l'aide `GRANT USAGE ON SCHEMA` de. Les utilisateurs administrateurs gèrent les objets dans le schéma public, tandis que les utilisateurs non administrateurs créent des objets dans des schémas créés par les utilisateurs. Le rôle d'administrateur peut s'octroyer n'importe quel autre rôle pour obtenir des autorisations sur des objets créés par l'utilisateur. Pour de plus amples informations, veuillez consulter Autoriser les rôles de la base de données à utiliser SQL dans votre base de données.
- Lorsque les pilotes appellent`PG_PREPARED_STATEMENTS`, Aurora DSQL fournit une vue à l'échelle du cluster des instructions préparées mises en cache. Vous pouvez voir plus d'instructions préparées par connexion que prévu pour le même cluster et le même rôle IAM. Aurora DSQL gère les noms des instructions de manière dynamique pendant la préparation.
- Lorsque vous vous connectez à partir d'instances IPv4 uniquement, assurez-vous que votre client est configuré pour les IPv4 connexions. Certains clients PostgreSQL tentent à la IPv4 fois de se connecter en mode IPv6 dualstack. Si la IPv4 connexion est limitée, le client peut tenter de renvoyer une `NetworkUnreachable` erreur sur IPv6 IPv4 les hôtes uniquement. Configurez votre client pour qu'il l'utilise IPv4 explicitement afin d'éviter ce comportement.
- Une fois qu'un utilisateur administrateur a créé un nouveau schéma `GRANT` et que les `REVOKE` modifications se sont propagées aux connexions existantes pendant la durée de vie de la

connexion (jusqu'à une heure). Pour un effet immédiat, établissez une nouvelle connexion après les modifications d'autorisation.

- Dans de rares scénarios de restauration de clusters liés multirégions, les opérations de restauration de clusters automatisées maintiennent une haute disponibilité, mais vous pouvez rencontrer des erreurs de contrôle de simultanéité ou de connexion transitoires. Dans la plupart des cas, seul un pourcentage de votre charge de travail est affecté. Lorsque vous rencontrez ces erreurs transitoires, réessayez votre transaction ou reconnectez-vous avec votre client.
- Certains clients SQL, tels que Datagrip, demandent des métadonnées système étendues pour renseigner les informations de schéma. Aurora DSQL fournit des métadonnées de base pour les fonctionnalités de requête SQL. L'affichage du schéma dans ces clients peut afficher des informations limitées par rapport à l'ensemble complet de leurs fonctionnalités.
- Pour vous assurer que les requêtes reconnaissent les schémas et les tables nouvellement créés, actualisez votre connexion après avoir créé ou supprimé des objets de base de données. Cela inclut les scénarios dans lesquels vous constatez `Schema Already Exists` des erreurs après la suppression d'un schéma ou lorsque vous interrogez des objets créés dans le cadre d'une autre connexion. Déconnectez-vous et reconnectez-vous, ou `SET search_path` réexécutez pour actualiser le cache du catalogue.
- Pour les requêtes complexes, utilisez-le `EXPLAIN ANALYZE VERBOSE` pour identifier les opérations à latence élevée et optimiser les plans de requêtes. La couverture des index permet de réduire considérablement les coûts du DPU en permettant des analyses d'index uniquement au lieu d'analyses de tables complètes. Pour de plus amples informations, veuillez consulter Utilisation des plans Aurora SQL EXPLAIN.
- Les limites de connexion sont gérées au niveau du cluster. Consultez Quotas de cluster pour demander des mises à jour des quotas.

Quotas de cluster et limites de base de données dans Amazon Aurora DSQL

Les sections suivantes décrivent les quotas de cluster et les limites de base de données pour Aurora DSQL.

Quotas de cluster

Vous Compte AWS disposez des quotas de cluster suivants dans Aurora DSQL. Pour demander une augmentation des quotas de service pour les clusters monorégionaux et multirégionaux au sein d'un cluster spécifique Région AWS, utilisez la page de console Service Quotas. Pour d'autres augmentations de quotas, contactez AWS Support.

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DSQL
Nombre maximal de clusters à région unique par Compte AWS	20 clusters	Oui	Code d'erreur d'API `ServiceQuotaExceededExcep`
Nombre maximal de clusters multirégionaux par Compte AWS	5 clusters	Oui	Code d'erreur d'API `ServiceQuotaExceededExcep`
Stockage maximum par cluster	Limite par défaut de 10 TiB, jusqu'à 256 TiB avec augmentat	Oui	`DISK_FULL(53100)`

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DSQL
	ion de limite approuvée		
Nombre maximal de connexions par cluster	10 000 connexions	Oui	`TOO_MANY_CONNECTIONS(53300)`
Taux de connexion maximum par cluster	100 connexions par seconde	Non	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacité de débordement de connexions maximale par cluster	1 000 connexions	Non	Aucun code d'erreur
Nombre maximal de tâches de restauration simultanées	4	Non	Aucun code d'erreur
Taux de remplissage de connexion	100 connexions par seconde	Non	Aucun code d'erreur

Limites de base de données dans Aurora DSQL

Le tableau suivant décrit les limites de la base de données dans Aurora DSQL.

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DS	Message d'erreur
Taille maximale combinée des colonnes utilisées dans une clé primaire	1 KIB	Non	54000	ERROR: key size too large
Taille combinée maximale des colonnes d'un index secondaire	1 KIB	Non	54000	ERROR: key size too large
Taille maximale d'une ligne dans une table	2 Mio	Non	54000	ERROR: maximum row size excee
Taille maximale d'une colonne qui ne fait pas partie d'un index	1 MiB	Non	54000	ERROR: maximum column size ex
Nombre maximal de colonnes dans une clé primaire ou un index secondaire	8	Non	54011	ERROR: more than 8 column key are not supported
Nombre maximal de colonnes dans une table	255	Non	54011	ERROR: tables can have at mos

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DS	Message d'erreur
Nombre maximal d'index dans une table	24	Non	54000	ERROR: more than 24 indexes p allowed
Taille maximale de toutes les données modifiées dans une transaction d'écriture	10 Mio	Non	54000	ERROR: transaction size limit DETAIL: Current transaction s 10mb
Nombre maximal de lignes de table pouvant être mutées dans un bloc de transacti on	3 000 lignes par transacti on. Consultez Considérations relatives à Aurora DSQL pour la compatibi lité avec PostgreSQL.	Non	54000	ERROR: transaction row limit
Quantité maximale de mémoire de base qu'une opération d'interrogation peut utiliser	128 MiO par transaction	Non	53200	ERROR: query requires too muc out of memory.

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DS	Message d'erreur
Nombre maximal de schémas définis dans une base de données	10	Non	54000	ERROR: more than 10 schemas n
Nombre maximal de tables dans une base de données	1 000 tables	Non	54000	ERROR: creating more than 100 allowed
Nombre maximal de bases de données dans un cluster	1	Non	Aucun code d'erreur	ERROR: unsupported statement
Durée de transaction maximale	5 minutes	Non	54000	ERROR: transaction age limit exceeded
Durée de connexion maximale	60 minutes	Non	Aucun code d'erreur	Aucun message d'erreur
Nombre maximal de vues dans une base de données	5 000	Non	54000	ERROR: creating more than 500 allowed
Taille maximale de la définition de la vue	2 Mio	Non	54000	ERROR: view definition too la

Description	Limite par défaut	Configurable ?	Code d'erreur Aurora DS	Message d'erreur
Nombre maximum de séquences	5 000	Non	`54000`	`ERROR: creating more than 500 not allowed`

Pour connaître les limites de type de données spécifiques à Aurora DSQL, consultez Types de données pris en charge dans Aurora DSQL.

Référence à l'API Aurora DSQL

Outre la AWS Management Console et l'AWS Command Line Interface (AWS CLI), Aurora DSQL fournit également une interface API. Vous pouvez utiliser les opérations d'API suivantes pour gérer vos ressources dans Aurora DSQL.

Pour obtenir la liste alphabétique des opérations d'API, consultez Actions.

Pour obtenir la liste alphabétique des types de données, consultez Types de données.

Pour consulter la liste des paramètres de requête courants, reportez-vous à la page Paramètres courants.

Pour obtenir la description des codes d'erreur, consultez Erreurs courantes.

Pour plus d'informations sur l'AWS CLI, consultez Référence de l'AWS Command Line Interface pour Aurora DSQL.

Résolution des problèmes dans Aurora DSQL

> **Note**
>
> Les rubriques suivantes fournissent des conseils de dépannage pour les erreurs et les problèmes que vous pouvez rencontrer en utilisant Aurora DSQL. Si vous rencontrez un problème qui n'est pas répertorié ici, contactez le support AWS

Rubriques

- Résolution des problèmes liés aux erreurs de connexion
- Résolution des erreurs liées à l'authentification
- Résolution des erreurs liées à l'autorisation
- Résolution des erreurs SQL
- Résolution des erreurs OCC
- Résolution des problèmes de SSL/TLS connexions

Résolution des problèmes liés aux erreurs de connexion

erreur : code d'erreur SSL non reconnu : 6 ou impossible d'accepter la connexion, le sni n'a pas été reçu

Vous utilisez peut-être une version de psql antérieure à la version 14, qui ne prend pas en charge l'indication du nom du serveur (SNI). Le SNI est requis lors de la connexion à Aurora DSQL.

Vous pouvez vérifier votre version client avec `psql --version`.

erreur : NetworkUnreachable

Une `NetworkUnreachable` erreur lors des tentatives de connexion peut indiquer que votre client ne prend pas en charge IPv6 les connexions, plutôt que de signaler un problème réseau réel. Cette erreur se produit généralement sur les instances IPv4 uniquement en raison de la façon dont les clients PostgreSQL gèrent les connexions à double pile. Lorsqu'un serveur prend en charge le mode double pile, ces clients résolvent d'abord les noms d'hôtes à la fois en adresses IPv4 et IPv6 en adresses. Ils tentent d'abord de se IPv4 connecter, puis essayent IPv6 si la connexion initiale échoue.

Si votre système n'est pas compatible IPv6, vous verrez une `NetworkUnreachable` erreur générale au lieu d'un message clair « IPv6 non pris en charge ».

Résolution des erreurs liées à l'authentification

L'authentification IAM a échoué pour l'utilisateur « ... »

Lorsque vous générez un jeton d'authentification IAM Aurora DSQL, la durée maximale que vous pouvez définir est d'une semaine. Au bout d'une semaine, vous ne pourrez plus vous authentifier avec ce jeton.

En outre, Aurora DSQL rejette votre demande de connexion si le rôle que vous avez endossé a expiré. Par exemple, si vous essayez de vous connecter avec un rôle IAM temporaire même si votre jeton d'authentification n'a pas expiré, Aurora DSQL rejettera la demande de connexion.

Pour plus d'informations sur le fonctionnement d'IAM avec Aurora DSQL, consultez Comprendre l'authentification et l'autorisation pour Aurora DSQL et Gestion des identités et des accès AWS dans Aurora DSQL.

Une erreur s'est produite (InvalidAccessKeyId) lors de l'appel de l' GetObjectopération : l'identifiant de clé d' AWS accès que vous avez fourni n'existe pas dans nos dossiers

IAM a rejeté votre demande. Pour plus d'informations, consultez Pourquoi les demandes sont-elles signées.

Le rôle IAM <role> n'existe pas

Aurora DSQL n'a pas pu trouver votre rôle IAM. Pour en savoir plus, consultez Rôles IAM.

Le rôle IAM doit ressembler à un ARN IAM

Voir Identifiants IAM - IAM ARNs pour plus d'informations.

Mauvais mappage utilisateur/action

Cette erreur se produit lorsque le type de jeton d'authentification ne correspond pas au rôle de base de données. Aurora DSQL utilise deux types de jetons : `DbConnectAdmin` pour le `admin` rôle et `DbConnect` pour les rôles de base de données personnalisés.

- Si vous voyez`Wrong user to action mapping. user: admin, action: DbConnect`, utilisez à la `generate-db-connect-admin-auth-token` place de`generate-db-connect-auth-token`.

- Si vous voyez`Wrong user to action mapping. user: myusername, action: DbConnectAdmin`, utilisez à la `generate-db-connect-auth-token` place de`generate-db-connect-admin-auth-token`.

Résolution des erreurs liées à l'autorisation

Rôle <role> non pris en charge

Aurora DSQL ne prend pas en charge cette opération `GRANT`. Voir Sous-ensembles de commandes SQL pris en charge dans Aurora DSQL.

Impossible d'établir une approbation avec le rôle <role>

Aurora DSQL ne prend pas en charge cette opération `GRANT`. Voir Sous-ensembles de commandes SQL pris en charge dans Aurora DSQL.

Le rôle <role> n'existe pas

Aurora DSQL n'a pas pu trouver l'utilisateur de base de données spécifié. Consultez Autoriser les rôles de base de données personnalisés à se connecter à un cluster.

ERREUR : autorisation refusée pour accorder l'approbation IAM avec le rôle <role>

Pour accorder l'accès à un rôle de base de données, vous devez être connecté à votre cluster avec le rôle d'administrateur. Pour plus d'informations, consultez Autoriser les rôles de base de données à utiliser SQL dans une base de données.

ERREUR : le rôle <role> doit avoir l'attribut LOGIN

Tous les rôles de base de données que vous créez doivent disposer de l'autorisation `LOGIN`.

Pour corriger cette erreur, assurez-vous que vous avez créé le rôle PostgreSQL avec l'autorisation `LOGIN`. Pour plus d'informations, consultez CREATE ROLE et ALTER ROLE dans la documentation de PostgreSQL.

ERREUR : le rôle <role> ne peut pas être supprimé, car certains objets en dépendent

Aurora DSQL renvoie une erreur si vous supprimez un rôle de base de données avec une relation IAM jusqu'à ce que vous révoquiez la relation en utilisant `AWS IAM REVOKE`. Pour plus d'informations, consultez Révocation de l'autorisation.

Résolution des erreurs SQL

Erreur : non pris en charge

Aurora DSQL ne prend pas en charge tous les dialectes basés sur PostgreSQL. Pour plus d'informations sur les fonctionnalités prises en charge, consultez Fonctionnalités PostgreSQL prises en charge dans Aurora DSQL.

Erreur : utiliser **`CREATE INDEX ASYNC`** à la place

Pour créer un index sur une table contenant des lignes existantes, vous devez utiliser la commande `CREATE INDEX ASYNC`. Pour plus d'informations, consultez Création d'index de manière asynchrone dans Aurora DSQL.

Résolution des erreurs OCC

OC000 « ERREUR : la mutation entre en conflit avec une autre transaction, réessayez si nécessaire »

Cette transaction a tenté de modifier les mêmes tuples qu'une autre transaction simultanée. Cela indique un conflit sur les tuples modifiés. Pour en savoir plus, reportez-vous à la section Contrôle de la simultanéité dans Aurora DSQL

OC001 « ERREUR : le schéma a été mis à jour par une autre transaction, réessayez si nécessaire »

Votre session PostgreSQL disposait d'une copie en cache du catalogue de schémas. Cette copie mise en cache était valide au moment du chargement. Appelons le moment T1 et la version V1.

Une autre transaction met à jour le catalogue au moment T2. Appelons cela V2.

Lorsque la session d'origine tente de lire les données stockées au moment T2, elle utilise toujours la version V1 du catalogue. La couche de stockage d'Aurora DSQL rejette la demande, car la dernière version du catalogue au moment T2 est V2.

Lorsque vous réessayez au moment T3 depuis la session d'origine, Aurora DSQL actualise le cache du catalogue. La transaction au moment T3 utilise le catalogue V2. Aurora DSQL terminera la transaction tant qu'aucune autre modification du catalogue n'aura été apportée depuis le moment T2.

Résolution des problèmes de SSL/TLS connexions

Erreur SSL : échec de la vérification du certificat

Cette erreur indique que le client ne peut pas vérifier le certificat du serveur. Assurez-vous que :

1. Le certificat Amazon Root CA 1 est correctement installé. Consultez Configuration des SSL/TLS certificats pour les connexions Aurora DSQL pour savoir comment valider et installer ce certificat.
2. La variable d'environnement `PGSSLROOTCERT` pointe vers le bon fichier de certificat.
3. Le fichier de certificat dispose des autorisations adéquates.

Code d'erreur SSL non reconnu : 6

Cette erreur se produit avec les clients PostgreSQL antérieurs à la version 14. Mettez à niveau votre client PostgreSQL vers la version 17 pour résoudre ce problème.

Erreur SSL : schéma non enregistré (Windows)

Il s'agit d'un problème connu lié au client Windows psql lors de l'utilisation de certificats système. Utilisez la méthode du fichier de certificat téléchargé décrite dans les instructions Connexion depuis Windows.

Fournir des commentaires sur Amazon Aurora DSQL

Si vous rencontrez des fonctionnalités essentielles pour votre migration mais qui ne sont pas actuellement prises en charge dans Aurora DSQL, AWS propose plusieurs canaux pour recueillir vos commentaires :

Canaux de feedback

Serveur Discord Aurora DSQL

Rejoignez le serveur Discord Aurora DSQL pour entrer en contact avec l'équipe et la communauté AWS. Partagez les demandes de fonctionnalités, discutez des défis liés à la migration et obtenez des commentaires en temps réel.

AWS Support

Si vous avez un plan de support AWS, créez un dossier de support pour discuter de vos exigences spécifiques et de vos besoins en matière de calendrier.

AWS Re:Post

Utilisez AWS Re:post pour poser des questions et partager des commentaires avec la communauté et les experts AWS.

Demandes de fonctionnalités efficaces

Lorsque vous demandez des fonctionnalités, fournissez :

- Description du cas d'utilisation : expliquez ce que vous essayez d'accomplir et pourquoi
- Solution actuelle : décrivez les alternatives que vous avez essayées
- Impact commercial : expliquez comment la fonctionnalité manquante affecte le calendrier de migration ou les fonctionnalités de votre application
- Niveau de priorité : indiquez si cela bloque votre migration ou s'il s'agit d'une nice-to-have amélioration

Historique de la documentation pour le guide de l'utilisateur Amazon Aurora DSQL

Le tableau suivant décrit les versions de la documentation d'Aurora DSQL.

Modification	Description	Date
Nouveau contenu : Connecteur Aurora DSQL pour .NET Npgsql	Ajout de la documentation pour le connecteur Aurora DSQL pour .NET Npgsql, qui intègre Npgsql à l'authentification IAM automatique. Le connecteur gère la génération de jetons, la configuration SSL et la gestion des connexions pour les applications .NET. Pour plus d'informations, consultez la section Connexion aux clusters Aurora DSQL avec un connecteur .NET Npgsql.	20 mars 2026
Contenu mis à jour : référence des commandes SQL et requêtes système	Ajout de START TRANSACTION et ROLLBACK à la référence des commandes de contrôle des transactions, avec END et ABORT comme alias. Ajout de requêtes système utiles pour récupérer les informations de version d'Aurora DSQL et de PostgreSQL. Pour plus d'informations, consultez le manuel de compatibilité de PostgreSQL.	13 mars 2026

Contenu mis à jour : chargement de données dans Aurora DSQL	Mise à jour du guide de chargement des données en tenant compte de `\copy` l'utilisation côté client, des meilleures pratiques INSERT et des conseils pour la pré-création de tables avant le chargement. Pour plus d'informations, consultez la section Chargement de données dans Aurora DSQL.	13 mars 2026
Nouveau contenu : Connecteur SQL Aurora pour Ruby pg	Ajout de la documentation pour le connecteur SQL Aurora pour Ruby pg, qui intègre l'authentification IAM automatique à la gem pg. Le connecteur gère la génération de jetons, la configuration SSL et la gestion des connexions pour les applications Ruby. Pour plus d'informations, voir Connexion aux clusters SQL Aurora à l'aide d'un connecteur Ruby pg.	12 mars 2026
Contenu mis à jour : tâches DDL asynchrones	Mise à jour de la `sys.jobs` documentation avec des détails supplémentaires sur la surveillance et la gestion des opérations DDL asynchrones. Pour plus d'informations, consultez le manuel de compatibilité de PostgreSQL.	6 mars 2026

Contenu mis à jour : génération de jetons d'authentification PHP	Ajout d'un onglet SDK PHP à la page de génération du jeton d'authentification. Pour plus d'informations, consultez la section Génération de jetons d'authentification.	5 mars 2026
Nouveau contenu : chargement de données dans Aurora DSQL	Ajout d'un guide pour le chargement de données dans des clusters Aurora DSQL, y compris l'utilisation de l'utilitaire de chargement Aurora DSQL. Pour plus d'informations, consultez la section Chargement de données dans Aurora DSQL.	5 mars 2026
Nouveau contenu : Séquences et colonnes d'identité	Ajout du support pour les séquences et les colonnes d'identité. Nouvelles pages de référence des commandes SQL pour les fonctions CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE et de manipulation de séquences. CREATE TABLE et ALTER TABLE ont été mis à jour pour inclure la syntaxe des colonnes d'identité. Ajout d'un nouveau guide pour choisir les types d'identifiants et les tailles de cache. Pour plus d'informations, consultez la section Séquences et colonnes d'identité.	11 février 2026

Nouveau contenu : Aurora DSQL Connector for Go	Ajout de la documentation pour le connecteur Aurora DSQL pour Go, qui intègre l'authentification IAM automatique à pgx. Le connecteur gère la génération de jetons, la configuration SSL et la gestion des connexions pour les applications Go. Pour plus d'informations, consultez la section Connexion aux clusters Aurora DSQL à l'aide d'un connecteur Go.	5 février 2026
Contenu mis à jour : outils de connectivité au cluster Amazon Aurora DSQL	Réorganisation de la documentation des outils de connectivité du cluster afin de clarifier la distinction entre les connecteurs AWS fournis, les adaptateurs et les outils tiers. Ajout de liens manquants vers des exemples de code. Pour plus d'informations, consultez les outils de connectivité au cluster Amazon Aurora DSQL.	26 janvier 2026

Nouveau contenu : Plug-in Aurora DSQL pour DBeaver Community Edition	Ajout de la documentation pour le plug-in Aurora DSQL pour DBeaver Community Edition, qui permet l'authentification IAM et simplifie la configuration des connexions pour les clusters Aurora DSQL. Comprend les instructions d'installation, la configuration de la connexion et les conseils de dépannage. Pour plus d'informations, consultez Utiliser DBeaver pour accéder à Aurora DSQL.	26 janvier 2026
Nouveau contenu : Aurora DSQL Driver pour SQLTools	Ajout de la documentation pour le pilote Aurora DSQL pour SQLTools, une extension Visual Studio Code qui permet aux développeurs de se connecter aux bases de données Aurora DSQL et de les interroger directement depuis VS Code grâce à l'authentification IAM automatique. Pour plus d'informations, consultez la section Utiliser le pilote Aurora DSQL pour SQLTools.	26 janvier 2026

Contenu mis à jour : Aurora SQL Steering : Skills and Powers	Ajout de documentation pour prendre en charge l'installation à l'aide de la CLI Skills pour un support indépendant de l'agent. La CLI Skills fournit une méthode de configuration simplifiée qui fonctionne avec plusieurs assistants de codage basés sur l'IA, notamment Claude Code, Cursor, Copilot, Gemini, etc. Pour plus d'informations, consultez Aurora DSQL Steering : Skills and Powers.	23 janvier 2026
Nouveau contenu : Adaptateur DSQL Aurora pour Tortoise ORM	Ajout du support pour Tortoise ORM, un framework ORM asynchrone en Python. L'adaptateur Aurora DSQL pour Tortoise ORM permet aux développeurs d'utiliser Tortoise ORM avec des clusters Aurora DSQL. Pour plus d'informations, consultez la section Adaptateurs et dialectes Aurora DSQL.	23 janvier 2026

Nouveau contenu : Aurora SQL Steering : Skills and Powers	Ajout d'une nouvelle documentation pour configurer le pilotage par IA avec Aurora DSQL à l'aide de compétences et de pouvoirs. Comprend des instructions de configuration pour Kiro Powers, Claude Skills, Gemini Skills et Codex Skills. Pour plus d'informations, consultez Aurora DSQL Steering : Skills and Powers.	16 janvier 2026
Support des index de types de données numériques	Ajout du support d'index pour le type de `numeric` données dans Aurora DSQL. Vous pouvez désormais utiliser `numeric` les colonnes comme clés primaires et dans les index secondaires. Pour plus d'informations, consultez la section Types de données pris en charge dans Aurora DSQL.	13 janvier 2026

Contenu mis à jour : sous-ensembles de commandes SQL pris en charge	Réorganisation de la documentation des commandes SQL en pages distinctes pour améliorer la navigation et la clarté. Chaque commande (CREATE TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW) possède désormais sa propre page dédiée. Pour plus d'informations, consultez la section Sous-ensembles de commandes SQL pris en charge.	6 janvier 2026
Contenu mis à jour : AWS Labs Aurora DSQL MCP Server	Mise à jour de la documentation du serveur MCP avec des approches d'installation détaillées pour Claude Code et Codex, y compris des exemples de fichiers d'installation et de configuration basés sur la CLI. Ajout de conseils complets pour rechercher les fichiers de configuration du client MCP dans différents outils de développement. Pour plus d'informations, consultez AWS Labs Aurora DSQL MCP Server.	19 décembre 2025

Contenu mis à jour : migration de PostgreSQL vers Aurora DSQL	La section sur la compatibilité avec PostgreSQL a été remaniée pour en faire un guide de migration complet. Comprend des informations sur la compatibilité du framework, des modèles de migration courants, des différences architecturales et des conseils de migration assistés par l'IA. Ajout d'un nouveau chapitre pour fournir des commentaires sur Aurora DSQL. Pour plus d'informations, consultez la section Migration de PostgreSQL vers Aurora DSQL.	16 décembre 2025
Contenu mis à jour : connexion à Aurora DSQL à l'aide de AWS PrivateLink	Ajout de documentation pour la configuration d'un DNS privé et de l'option de connexion par identifiant de cluster pour aider les clients utilisant le AWS PrivateLink Direct Connect peering avec ou Amazon VPC. Comprend des instructions pour se connecter sans DNS privé à l'aide de l'option de `amzn-cluster-id` connexion. Pour plus d'informations, consultez la section Gestion et connexion aux clusters Aurora DSQL à l'aide AWS PrivateLink de.	11 décembre 2025

Contenu mis à jour : cycle de vie du cluster Aurora DSQL	Documentation mise à jour pour la gestion du cycle de vie des clusters Aurora DSQL. Explique les définitions de l'état du cluster, les transitio ns d'état et les opérations disponibles pendant les états inactif et inactif. Pour plus d'informations, consultez la section Cycle de vie du cluster Aurora DSQL.	4 décembre 2025
Nouveau contenu : connecteu rs Aurora DSQL pour Python et Node.js	Ajout de documentation pour les connecteurs SQL Aurora pour Python (psycopg, psycopg2, asyncpg) et Node.js (node-postgres, Postgres.js). Ces connecteurs intègrent l'authentification IAM pour connecter les applications aux clusters Aurora DSQL. Pour plus d'informations, consultez la section Connecteurs Aurora DSQL.	21 novembre 2025

Nouveau contenu : Utilisation JupyterLab avec Aurora DSQL	Ajout d'un step-by-step guide pour la connexion et l'interrogation d'Aurora DSQL à l'aide de JupyterLab Python. Comprend des instructions pour les JupyterLab installations locales et les environnements Amazon SageMaker AI. Pour plus d'informations, consultez la section Utilisation JupyterLab avec Aurora DSQL.	20 novembre 2025
Contenu mis à jour : Quotas pour Aurora DSQL	Le quota de stockage maximal du cluster a été mis à jour, passant de 128 TiB à 256 TiB. Pour plus d'informations, consultez la section Quotas pour Aurora DSQL.	19 novembre 2025
Nouveau contenu : prise en main de l'éditeur de requêtes SQL Aurora	Ajout de documentation sur l'utilisation de l'éditeur de requêtes SQL Aurora dans la console AWS de gestion. Inclut les prérequis, la configuration de la connexion et les instructions d'exécution des requêtes. Pour plus d'informations, voir Mise en route avec l'éditeur de requêtes SQL Aurora.	18 novembre 2025

Support des politiques basées sur les ressources pour Amazon Aurora DSQL	Ajout de la prise en charge des politiques basées sur les ressources (RBP) avec de nouvelles autorisat ions :`PutClusterPolicy` , et`GetClusterPolicy` . `DeleteClusterPolicy` Ces autorisations permetten t de gérer les politiques en ligne associées aux clusters Aurora DSQL pour un contrôle d'accès précis. Politiques gérées mises à jour : AmazonAurora DSQLFull accès AmazonAur ora DSQLReadOnlyAccess, et AmazonAurora DSQLConso le FullAccess pour inclure les fonctionnalités RBP. Pour plus d'informations, consultez les politiques AWS gérées pour Amazon Aurora DSQL.	15 octobre 2025
Connecteur JDBC Aurora DSQL	Ajout de documentation pour le connecteur JDBC Aurora DSQL, un connecteur pgjDBC qui intègre l'authentification IAM pour connecter des applications Java aux clusters Amazon Aurora DSQL. Pour plus d'informations, voir Connexion aux clusters Aurora DSQL à l'aide d'un connecteur JDBC.	2 septembre 2025

AWS mises à jour des politiques gérées pour AWS FIS l'intégration	`AmazonAuroraDSQLConsoleFullAccess` Politiques mises à jour `AmazonAuroraDSQLFullAccess` et prises en charge de AWS Fault Injection Service l'intégration avec Aurora DSQL. Cela vous permet d'injecter des défaillances dans des clusters Aurora DSQL à une seule région ou multi-régions afin de tester la tolérance aux pannes de vos applications. Pour en savoir plus sur ces politiques, consultez Mises à jour des politiques gérées par AWS.	19 août 2025
Disponibilité générale (GA) d'Amazon Aurora DSQL	Amazon Aurora DSQL est désormais généralement disponible avec un support supplémentaire pour la CloudWatch surveillance, des fonctionnalités de protection des données améliorées et AWS Backup l'intégration. Pour plus d'informations, consultez les sections Surveillance d'Aurora DSQL avec CloudWatch, Backup et restauration pour Amazon Aurora DSQL et Chiffrement des données pour Amazon Aurora DSQL.	27 mai 2025

AmazonAuroraDSQLFullMise à jour des accès	Permet d'effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, y compris le démarrage, l'arrêt et la surveillance des tâches. Permet également d'utiliser des clés KMS gérées par le client pour le chiffrement des clusters. Pour plus d'informations, voir AmazonAuroraDSQLFullAccès et utilisation des rôles liés à un service dans Aurora DSQL.	21 mai 2025
AmazonAuroraDSQLConsoleFullAccess update	Permet d'effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL via l'AWS Console Home. Cela inclut le démarrage, l'arrêt et le suivi des tâches. Cela prend également en charge l'utilisation de clés KMS gérées par le client pour le chiffrement des clusters et le lancement d' AWS CloudShell. Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLConsoleFullAccessde rôles liés à un service dans Aurora DSQL.	21 mai 2025

AmazonAuroraDSQLReadOnlyAccess update	Inclut la possibilité de déterminer le nom de service de point de terminaison VPC correct lors de la connexion à vos clusters Aurora DSQL via AWS PrivateLink Aurora DSQL. Cela crée des points de terminaison uniques par cellule. Cette API vous permet donc d'identifier le point de terminaison approprié pour votre cluster et d'éviter les erreurs de connexion. Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLReadOnlyAccessde rôles liés à un service dans Aurora DSQL.	13 mai 2025

AmazonAuroraDSQLFullMise à jour des accès	La politique ajoute quatre nouvelles autorisations pour créer et gérer des clusters de bases de données sur plusieurs Régions AWS : `PutMultiRegionProperties` `PutWitnessRegion` ,`AddPeerCluster` , et`RemovePeerCluster` . Ces autorisations incluent des contrôles au niveau des ressources et des clés de condition afin que vous puissiez contrôler les utilisateurs des clusters que vous pouvez modifier. La politique ajoute également l'`GetVpcEndpointServiceName` autorisation de vous aider à vous connecter à vos clusters Aurora DSQL via AWS PrivateLink. Pour plus d'informations, reportez-vous à la section Utilisation AmazonAuroraDSQLConsoleFullAccessde rôles liés à un service dans Aurora DSQL.	13 mai 2025

AmazonAuroraDSQLConsoleFullAccess update	Ajout de nouvelles autorisations à Aurora DSQL pour prendre en charge la gestion de clusters multi-régions et la connexion des points de terminaison d'un VPC. Les nouvelles autorisations incluent : `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` . Consultez AmazonAuroraDSQLConsoleFullAccessset Utilisation des rôles liés à un service dans Aurora DSQL.	13 mai 2025
AuroraDsqlServiceLinkedRolePolicy update	Permet de publier des métriques sur les espaces de noms `AWS/AuroraDSQL` et `AWS/Usage` `CloudWatch` dans la politique. Cela permet au service ou au rôle associé d'émettre des données d'utilisation et de performance plus complètes CloudWatch dans votre environnement. Pour plus d'informations, reportez-vous à la section Utilisation AuroraDsqlServiceLinkedRolePolicyde rôles liés à un service dans Aurora DSQL.	8 mai 2025

AWS PrivateLink pour Amazon Aurora DSQL	Aurora DSQL prend désormais en charge AWS PrivateLink. Vous pouvez ainsi simplifier la connectivité réseau privé entre les clouds privés virtuels (VPCs), Aurora DSQL et vos centres de données sur site à l'aide de l'interface Amazon VPC, des points de terminaison et des adresses IP privées. AWS PrivateLink Pour plus d’informations, consultez Gestion et connexion aux clusters Amazon Aurora DSQL à l’aide d’ AWS PrivateLink.	8 mai 2025
Première version	Version initiale du Guide de l’utilisateur Amazon Aurora DSQL.	3 décembre 2024

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.